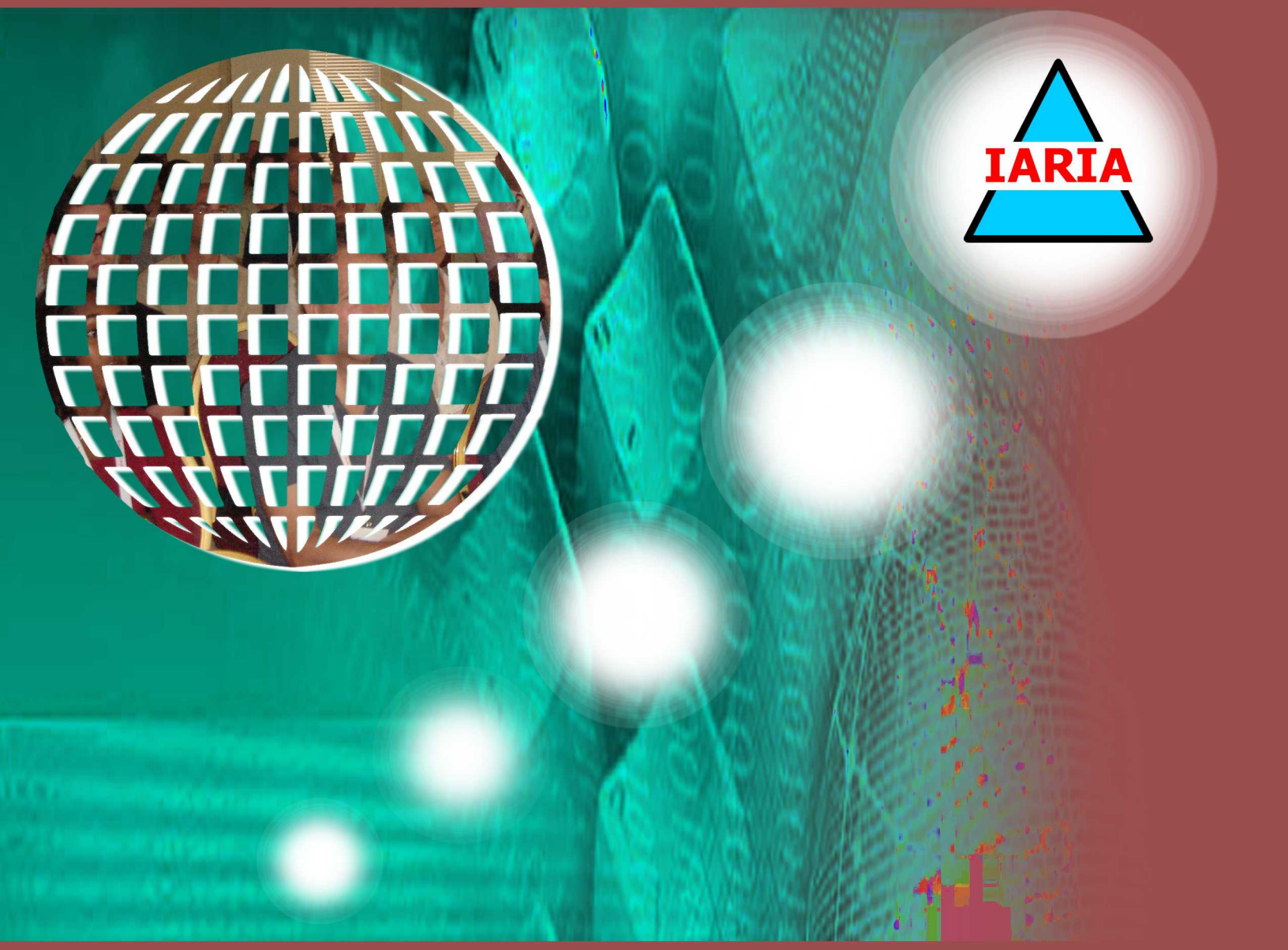
International Journal on Advances in Software



2014 vol. 7 nr. 1&2

The *International Journal on Advances in Software* is published by IARIA.
ISSN: 1942-2628
journals site: http://www.iariajournals.org
contact: petre@iaria.org

Sponsored by IARIA
www.iaria.org

International Journal on Advances in Software
Volume 7, Number 1 & 2, 2014

Kenneth Duncan Boness, University of Reading, England
Fernando Boronat Seguí, Universidad Politecnica de Valencia, Spain
Pierre Borne, Ecole Centrale de Lille, France
Farid Bourennani, University of Ontario Institute of Technology (UOIT), Canada
Narhimene Boustia, Saad Dahlab University - Blida, Algeria
Hongyu Pei Breivold, ABB Corporate Research, Sweden
Carsten Brockmann, Universität Potsdam, Germany
Mikey Browne, IBM, USA
Antonio Bucchiarone, Fondazione Bruno Kessler, Italy
Georg Buchgeher, Software Competence Center Hagenberg GmbH, Austria
Dumitru Burdescu, University of Craiova, Romania
Martine Cadot, University of Nancy / LORIA, France
Isabel Candal-Vicente, Universidad del Este, Puerto Rico
Juan-Vicente Capella-Hernández, Universitat Politècnica de València, Spain
Jose Carlos Metrolho, Polytechnic Institute of Castelo Branco, Portugal
Alain Casali, Aix-Marseille University, France
Alexandra Suzana Cernian, University POLITEHNICA of Bucharest, Romania
Yaser Chaaban, Leibniz University of Hanover, Germany
Savvas A. Chatzichristofis, Democritus University of Thrace, Greece
Antonin Chazalet, Orange, France
Jiann-Liang Chen, National Dong Hwa University, China
Shiping Chen, CSIRO ICT Centre, Australia
Wen-Shiung Chen, National Chi Nan University, Taiwan
Zhe Chen, College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China PR
Po-Hsun Cheng, National Kaohsiung Normal University, Taiwan
Yoonsik Cheon, The University of Texas at El Paso, USA
Lau Cheuk Lung, INE/UFSC, Brazil
Robert Chew, Lien Centre for Social Innovation, Singapore
Andrew Connor, Auckland University of Technology, New Zealand
Rebeca Cortázar, University of Deusto, Spain
Noël Crespi, Institut Telecom, Telecom SudParis, France
Carlos E. Cuesta, Rey Juan Carlos University, Spain
Duilio Curcio, University of Calabria, Italy
Mirela Danubianu, "Stefan cel Mare" University of Suceava, Romania
Paulo Asterio de Castro Guerra, Tapijara Programação de Sistemas Ltda. - Lambari, Brazil
Cláudio de Souza Baptista, University of Campina Grande, Brazil
Maria del Pilar Angeles, Universidad Nacional Autonónoma de México, México
Rafael del Vado Vírseda, Universidad Complutense de Madrid, Spain
Giovanni Denaro, University of Milano-Bicocca, Italy
Hepu Deng, RMIT University, Australia
Nirmit Desai, IBM Research, India
Vincenzo Deufemia, Università di Salerno, Italy
Leandro Dias da Silva, Universidade Federal de Alagoas, Brazil
Javier Diaz, Rutgers University, USA
Nicholas John Dingle, University of Manchester, UK

Roland Dodd, CQUniversity, Australia
Aijuan Dong, Hood College, USA
Suzana Dragicevic, Simon Fraser University- Burnaby, Canada
Cédric du Mouza, CNAM, France
Ann Dunkin, Palo Alto Unified School District, USA
Jana Dvorakova, Comenius University, Slovakia
Lars Ebrecht, German Aerospace Center (DLR), Germany
Hans-Dieter Ehrich, Technische Universität Braunschweig, Germany
Jorge Ejarque, Barcelona Supercomputing Center, Spain
Atilla Elçi, Aksaray University, Turkey
Khaled El-Fakih, American University of Sharjah, UAE
Gledson Elias, Federal University of Paraíba, Brazil
Sameh Elnikety, Microsoft Research, USA
Fausto Fasano, University of Molise, Italy
Michael Felderer, University of Innsbruck, Austria
João M. Fernandes, Universidade de Minho, Portugal
Luis Fernandez-Sanz, University of de Alcala, Spain
Felipe Ferraz, C.E.S.A.R, Brazil
Adina Magda Florea, University "Politehnica" of Bucharest, Romania
Wolfgang Fohl, Hamburg Universiy, Germany
Simon Fong, University of Macau, Macau SAR
Gianluca Franchino, Scuola Superiore Sant'Anna, Pisa, Italy
Naoki Fukuta, Shizuoka University, Japan
Martin Gaedke, Chemnitz University of Technology, Germany
Félix J. García Clemente, University of Murcia, Spain
José García-Fanjul, University of Oviedo, Spain
Felipe Garcia-Sanchez, Universidad Politecnica de Cartagena (UPCT), Spain
Michael Gebhart, Gebhart Quality Analysis (QA) 82, Germany
Tejas R. Gandhi, Virtua Health-Marlton, USA
Andrea Giachetti, Università degli Studi di Verona, Italy
Robert L. Glass, Griffith University, Australia
Afzal Godil, National Institute of Standards and Technology, USA
Luis Gomes, Universidade Nova Lisboa, Portugal
Diego Gonzalez Aguilera, University of Salamanca - Avila, Spain
Pascual Gonzalez, University of Castilla-La Mancha, Spain
Björn Gottfried, University of Bremen, Germany
Victor Govindaswamy, Texas A&M University, USA
Gregor Grambow, University of Ulm, Germany
Carlos Granell, European Commission / Joint Research Centre, Italy
Christoph Grimm, University of Kaiserslautern, Austria
Michael Grottke, University of Erlangen-Nuernberg, Germany
Vic Grout, Glyndwr University, UK
Ensar Gul, Marmara University, Turkey
Richard Gunstone, Bournemouth University, UK
Zhensheng Guo, Siemens AG, Germany
Phuong H. Ha, University of Tromso, Norway

Ismail Hababeh, German Jordanian University, Jordan
Shahliza Abd Halim, Lecturer in Universiti Teknologi Malaysia, Malaysia
Herman Hartmann, University of Groningen, The Netherlands
Jameleddine Hassine, King Fahd University of Petroleum & Mineral (KFUPM), Saudi Arabia
Tzung-Pei Hong, National University of Kaohsiung, Taiwan
Peizhao Hu, NICTA, Australia
Chih-Cheng Hung, Southern Polytechnic State University, USA
Edward Hung, Hong Kong Polytechnic University, Hong Kong
Noraini Ibrahim, Universiti Teknologi Malaysia, Malaysia
Anca Daniela Ionita, University "POLITEHNICA" of Bucharest, Romania
Chris Ireland, Open University, UK
Kyoko Iwasawa, Takushoku University - Tokyo, Japan
Mehrshid Javanbakht, Azad University - Tehran, Iran
Wassim Jaziri, ISIM Sfax, Tunisia
Dayang Norhayati Abang Jawawi, Universiti Teknologi Malaysia (UTM), Malaysia
Jinyuan Jia, Tongji University. Shanghai, China
Maria Joao Ferreira, Universidade Portucalense, Portugal
Ahmed Kamel, Concordia College, Moorhead, Minnesota, USA
Teemu Kanstrén, VTT Technical Research Centre of Finland, Finland
Nittaya Kerdprasop, Suranaree University of Technology, Thailand
Ayad ali Keshlaf, Newcastle University, UK
Nhien An Le Khac, University College Dublin, Ireland
Sadegh Kharazmi, RMIT University - Melbourne, Australia
Kyoung-Sook Kim, National Institute of Information and Communications Technology, Japan
Youngjae Kim, Oak Ridge National Laboratory, USA
Roger "Buzz" King, University of Colorado at Boulder, USA
Cornel Klein, Siemens AG, Germany
Alexander Knapp, University of Augsburg, Germany
Radek Koci, Brno University of Technology, Czech Republic
Christian Kop, University of Klagenfurt, Austria
Michal Krátký, VŠB - Technical University of Ostrava, Czech Republic
Narayanan Kulathuramaiyer, Universiti Malaysia Sarawak, Malaysia
Satoshi Kurihara, Osaka University, Japan
Eugenijus Kurilovas, Vilnius University, Lithuania
Philippe Lahire, Université de Nice Sophia-Antipolis, France
Alla Lake, Linfo Systems, LLC, USA
Fritz Laux, Reutlingen University, Germany
Luigi Lavazza, Università dell'Insubria, Italy
Fábio Luiz Leite Júnior, Universidade Estadual da Paraiba,Brazil
Alain Lelu, University of Franche-Comté / LORIA, France
Cynthia Y. Lester, Georgia Perimeter College, USA
Clement Leung, Hong Kong Baptist University, Hong Kong
Weidong Li, University of Connecticut, USA
Corrado Loglisci, University of Bari, Italy
Francesco Longo, University of Calabria, Italy
Sérgio F. Lopes, University of Minho, Portugal

Pericles Loucopoulos, Loughborough University, UK
Alen Lovrencic, University of Zagreb, Croatia
Qifeng Lu, MacroSys, LLC, USA
Xun Luo, Qualcomm Inc., USA
Shuai Ma, Beihang University, China
Stephane Maag, Telecom SudParis, France
Ricardo J. Machado, University of Minho, Portugal
Maryam Tayefeh Mahmoudi, Research Institute for ICT, Iran
Nicos Malevris, Athens University of Economics and Business, Greece
Herwig Mannaert, University of Antwerp, Belgium
José Manuel Molina López, Universidad Carlos III de Madrid, Spain
Francesco Marcelloni, University of Pisa, Italy
Eda Marchetti, Consiglio Nazionale delle Ricerche (CNR), Italy
Leonardo Mariani, University of Milano Bicocca, Italy
Gerasimos Marketos, University of Piraeus, Greece
Abel Marrero, Bombardier Transportation, Germany
Adriana Martin, Universidad Nacional de la Patagonia Austral / Universidad Nacional del Comahue, Argentina
Goran Martinovic, J.J. Strossmayer University of Osijek, Croatia
Paulo Martins, University of Trás-os-Montes e Alto Douro (UTAD), Portugal
Stephan Mäs, Technical University of Dresden, Germany
Constandinos Mavromoustakis, University of Nicosia, Cyprus
Jose Merseguer, Universidad de Zaragoza, Spain
Seyedeh Leili Mirtaheri, Iran University of Science & Technology, Iran
Lars Moench, University of Hagen, Germany
Yasuhiko Morimoto, Hiroshima University, Japan
Muhanna A Muhanna, University of Nevada - Reno, USA
Antonio Navarro Martín, Universidad Complutense de Madrid, Spain
Filippo Neri, University of Naples, Italy
Toàn Nguyên, INRIA Grenobel Rhone-Alpes/ Montbonnot, France
Muaz A. Niazi, Bahria University, Islamabad, Pakistan
Natalja Nikitina, KTH Royal Institute of Technology, Sweden
Marcellin Julius Nkenlifack, Université de Dschang, Cameroun
Roy Oberhauser, Aalen University, Germany
Pablo Oliveira Antonino, Fraunhofer IESE, Germany
Rocco Oliveto, University of Molise, Italy
Sascha Opletal, Universität Stuttgart, Germany
Flavio Oquendo, European University of Brittany/IRISA-UBS, France
Claus Pahl, Dublin City University, Ireland
Marcos Palacios, University of Oviedo, Spain
Constantin Paleologu, University Politehnica of Bucharest, Romania
Kai Pan, UNC Charlotte, USA
Yiannis Papadopoulos, University of Hull, UK
Andreas Papasalouros, University of the Aegean, Greece
Rodrigo Paredes, Universidad de Talca, Chile
Päivi Parviainen, VTT Technical Research Centre, Finland
João Pascoal Faria, Faculty of Engineering of University of Porto / INESC TEC, Portugal

Fabrizio Pastore, University of Milano - Bicocca, Italy
Kunal Patel, Ingenuity Systems, USA
Óscar Pereira, Instituto de Telecomunicacoes - University of Aveiro, Portugal
Willy Picard, Poznań University of Economics, Poland
Jose R. Pires Manso, University of Beira Interior, Portugal
Sören Pirk, Universität Konstanz, Germany
Meikel Poess, Oracle Corporation, USA
Thomas E. Potok, Oak Ridge National Laboratory, USA
Dilip K. Prasad, Nanyang Technological University, Singapore
Christian Prehofer, Fraunhofer-Einrichtung für Systeme der Kommunikationstechnik ESK, Germany
Ela Pustułka-Hunt, Bundesamt für Statistik, Neuchâtel, Switzerland
Mengyu Qiao, South Dakota School of Mines and Technology, USA
Kornelije Rabuzin, University of Zagreb, Croatia
J. Javier Rainer Granados, Universidad Politécnica de Madrid, Spain
Muthu Ramachandran, Leeds Metropolitan University, UK
Thurasamy Ramayah, Universiti Sains Malaysia, Malaysia
Prakash Ranganathan, University of North Dakota, USA
José Raúl Romero, University of Córdoba, Spain
Henrique Rebêlo, Federal University of Pernambuco, Brazil
Bernd Resch, Massachusetts Institute of Technology, USA
Hassan Reza, UND Aerospace, USA
Elvinia Riccobene, Università degli Studi di Milano, Italy
Daniel Riesco, Universidad Nacional de San Luis, Argentina
Mathieu Roche, LIRMM / CNRS / Univ. Montpellier 2, France
Aitor Rodríguez-Alsina, University Autonoma of Barcelona, Spain
José Rouillard, University of Lille, France
Siegfried Rouvrais, TELECOM Bretagne, France
Claus-Peter Rückemann, Leibniz Universität Hannover / Westfälische Wilhelms-Universität Münster / North-German Supercomputing Alliance, Germany
Djamel Sadok, Universidade Federal de Pernambuco, Brazil
Arun Saha, Fujitsu, USA
Ismael Sanz, Universitat Jaume I, Spain
M. Saravanan, Ericsson India Pvt. Ltd -Tamil Nadu, India
Idrissa Sarr, University of Cheikh Anta Diop, Dakar, Senegal / University of Quebec, Canada
Patrizia Scandurra, University of Bergamo, Italy
Giuseppe Scanniello, Università degli Studi della Basilicata, Italy
Daniel Schall, Vienna University of Technology, Austria
Rainer Schmidt, Munich University of Applied Sciences, Germany
Cristina Seceleanu, Mälardalen University, Sweden
Sebastian Senge, TU Dortmund, Germany
Isabel Seruca, Universidade Portucalense - Porto, Portugal
Kewei Sha, Oklahoma City University, USA
Simeon Simoff, University of Western Sydney, Australia
Jacques Simonin, Institut Telecom / Telecom Bretagne, France
Cosmin Stoica Spahiu, University of Craiova, Romania
George Spanoudakis, City University London, UK

Alin Stefanescu, University of Pitesti, Romania
Lena Strömbäck, SMHI, Sweden
Kenji Suzuki, The University of Chicago, USA
Osamu Takaki, Japan Advanced Institute of Science and Technology, Japan
Antonio J. Tallón-Ballesteros, University of Seville, Spain
Wasif Tanveer, University of Engineering & Technology - Lahore, Pakistan
Ergin Tari, Istanbul Technical University, Turkey
Steffen Thiel, Furtwangen University of Applied Sciences, Germany
Jean-Claude Thill, Univ. of North Carolina at Charlotte, USA
Pierre Tiako, Langston University, USA
Ioan Toma, STI, Austria
Božo Tomas, HT Mostar, Bosnia and Herzegovina
Davide Tosi, Università degli Studi dell'Insubria, Italy
Peter Trapp, Ingolstadt, Germany
Guglielmo Trentin, National Research Council, Italy
Dragos Truscan, Åbo Akademi University, Finland
Chrisa Tsinaraki, Technical University of Crete, Greece
Roland Ukor, FirstLinq Limited, UK
Torsten Ullrich, Fraunhofer Austria Research GmbH, Austria
José Valente de Oliveira, Universidade do Algarve, Portugal
Dieter Van Nuffel, University of Antwerp, Belgium
Shirshu Varma, Indian Institute of Information Technology, Allahabad, India
Konstantina Vassilopoulou, Harokopio University of Athens, Greece
Miroslav Velev, Aries Design Automation, USA
Tanja E. J. Vos, Universidad Politécnica de Valencia, Spain
Krzysztof Walczak, Poznan University of Economics, Poland
Jianwu Wang, San Diego Supercomputer Center / University of California, San Diego, USA
Yandong Wang, Wuhan University, China
Rainer Weinreich, Johannes Kepler University Linz, Austria
Stefan Wesarg, Fraunhofer IGD, Germany
Sebastian Wieczorek, SAP Research Center Darmstadt, Germany
Wojciech Wiza, Poznan University of Economics, Poland
Martin Wojtczyk, Technische Universität München, Germany
Hao Wu, School of Information Science and Engineering, Yunnan University, China
Mudasser F. Wyne, National University, USA
Zhengchuan Xu, Fudan University, P.R.China
Yiping Yao, National University of Defense Technology, Changsha, Hunan, China
Stoyan Yordanov Garbatov, Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento, INESC-ID, Portugal
Weihai Yu, University of Tromsø, Norway
Wenbing Zhao, Cleveland State University, USA
Hong Zhu, Oxford Brookes University, UK
Qiang Zhu, The University of Michigan - Dearborn, USA

International Journal on Advances in Software
Volume 7, Numbers 1 & 2, 2014

CONTENTS

Multiagent Genetic Optimisation to Solve the Project Scheduling Problem under Uncertainty

Konstantin Aksyonov and Anna Antonova
Department of Information Technologies
Ural Federal University, UrFU
Yekaterinburg, Russia
wiper99@mail.ru, antonovaannas@gmail.com

***Abstract*—This paper considers a project scheduling problem under uncertainty, which belongs to a class of multiobjective problems of complex systems control whose decision search time grows exponentially depending on the problem dimension. In this paper, we propose a multiagent genetic optimisation method based on evolutionary and multiagent modelling by implementing different decision searching strategies, including a simulation module and numerical methods application. The comparative analysis of the scheduling methods has shown that the proposed method supports all features that might be useful in effective decision searching of the stochastic scheduling problem. The proposed multiagent genetic optimisation method, the MS Project resource reallocation method, and a heuristic simulation method were compared whilst addressing a real-world deterministic scheduling problem. The comparison has shown: firstly, the unsuitability of the MS Project planning method for solving the formulated problem; and secondly, both the advantage of the multiagent genetic optimisation method in terms of economic effect and disadvantage in terms of performance. Experimental results in conditions of uncertainty demonstrate the effectiveness of the proposed method. Some techniques to reduce the impact of the method's disadvantage are proposed in the conclusion, as well as the aims of future work.**

Keywords-project scheduling; genetic algorithms; simulation; subcontract work optimisation; problem under uncertainty.

I. INTRODUCTION

This paper is an improved and expanded version of the ICCGI 2013 conference paper "Multiagent Genetic Optimisation to Solve the Project Scheduling Problem" [1]. The paper extends the scheduling method proposed in the original paper by taking into account environment uncertainty removal with the help of the integration of numerical methods, simulation, multiagent, and evolutionary modelling. A comparison of the new method and existing scheduling methods is conducted in this paper. An application of the new method to a real scheduling problem is described.

The scheduling problem is one of the key problems in the management of organisational and technical systems. Inefficient scheduling can lead to financial losses, quality of service losses, and loss of competitiveness for the company. Companies with various different scopes are faced with the scheduling problem, for example, industrial and project companies, shopping centres, hospitals, and call centres.

There are several types of scheduling problem depending on the application sphere: operations calendar planning [2]–[6], assignment of limited resources to a set of tasks [7]–[9], and the travelling salesman problem [10].

Classical scheduling problem-solving methods have a number of disadvantages. Thus, the use of combinatorial methods and mathematical programming is associated with internal difficulties because the model of system processes is nonlinear, non-convex, and non-differentiable [11]. In addition, these methods are applied poorly to problems with dynamically changing constraints. Simulation takes into account the dynamic nature of the problem, but leads to a random search process, which does not guarantee optimal decision finding. The use of genetic optimisation allows the shortcomings of the previous methods to be overcome [10]. The application of genetic optimisation to the scheduling problem with defined constraints is widely considered in the literature [2]–[10].

In the real world, the scheduling problem is connected to the uncertainty of environment behaviour and is a stochastic version of the classical scheduling problem. It can involve many sources of uncertainty: activity duration, renewable resource availability, resource consumption, and cost of activity [11]–[16]. Mainly non-structural (parametric) uncertainty is introduced into the basic deterministic scheduling problem by researchers [17]. The design of the optimal (efficient) calendar work plan taking into account the structural uncertainty associated with the insertion of new projects is a topical task.

This paper focuses on the project scheduling problem under conditions of structural uncertainty by using evolutionary computation [18], simulation, and numerical methods of uncertainty removal [19]. The remainder of the paper is organised as follows. Section II provides an overview of the related works in the field of deterministic and stochastic scheduling. Section III formulates the deterministic project scheduling problem with time constraint. Section IV introduces the genetic algorithm based on an annealing simulation and novelty search. Section V describes a dynamic model of multiagent resource conversion processes that has been selected as a system formalisation model. Section VI presents the algorithm for the multiagent genetic optimisation program based on the

integration of evolutionary computation and multiagent simulation. Section VII introduces the multiagent genetic optimisation method under uncertainty. Section VIII presents the algorithm of the multiagent genetic optimisation program under uncertainty based on the integration of the evolutionary computation, multiagent simulation, and numerical optimisation methods. Section IX presents a comparative analysis of the existing methods and the proposed method of solving the deterministic and stochastic scheduling problem. Section X evaluates the practical implementation of the multiagent genetic optimisation program to solve a real-world scheduling problem, both deterministic and stochastic. Section XI concludes this paper and explores future work.

II. Related Work

In general, the deterministic scheduling problem is connected to the problem of seeking an operations sequence that satisfies the constraints and optimises the objective functions. Renewable resources (such as staff or equipment) are usually considered when studying the scheduling problem. For certain tasks (for example, production planning) nonrenewable resources should be determined [2].

In the various scheduling problem studies different constraint sets are considered, depending on the specific task. Four constraint types were identified in [4]: resource, precedence, physical layout, and information constraints. The time constraint type should be added to the list of constraint types when analysing workflow inside a project development company. Time limitation is associated with having a time frame for the operations start date.

All constraints, except precedence ones, have been studied by Brezuliani et al. [7]. Precedence and resource constraints were considered by Okada et al. [2], Klimek [3], Abdel-Khalek et al. [5], and Dhingra and Chandna [8]. Resource and information constraints were studied by Yang and Wu [9]. Resource, precedence, and time constraints were considered by Karova et al. [6]. A study of scheduling with a resource constraint to determine a public transport route was presented by Osaba et al. [10].

The optimisation objects are different in the studies reviewed. The classical objective function of working time (makespan) minimization was considered by Sriprasert and Dawood [4], Osaba et al. [10], He and Wan [12], Zhang and Chen [13], Csebfalvi [14], and Artigues et al. [15]. The objective function of constraints violation penalty minimization has been considered by Karova et al. [6] and Yang and Wu [9]. Both mentioned objective functions were considered by Okada et al. [2], Brezuliani et al. [7], and Dhingra and Chandna [8]. The objective function of net present value of discounted cash flow maximization was considered by Chen and Zhang [16].

There are different ways of conducting an objective function evaluation: analytical methods, simulation, artificial neural networks, fuzzy systems, and component modelling. Analytical methods are the most widely used; the drawback of this approach is the lack of analysis of the dynamic behaviour of complex systems. This drawback is overcome by using the simulation model of Osaba et al. [10] to evaluate the objective function. The integration of evolutionary modelling and simulation can limit the random search space and enhance heuristic optimisation by taking into account the dynamically changing constraints of the scheduling problem.

The reviewed studies do not consider subcontracted workforce optimisation, while this problem is very real to developers and even to mass production enterprises. The optimisation problem of the subcontracted workforce is connected to scheduling subcontractors in order to maximise the utilisation of the company's own resources. In the literature, a problem regarding the appropriate selection of subcontractors using artificial intelligence methods was studied by Chen et al. [20]. A subcontract optimisation technique based on a simulation and heuristics has been suggested by Aksyonov and Antonova [21]. The current article considers new subcontract optimisation techniques for a deterministic scheduling problem with the use of a genetic algorithm.

An unexpected external influence may result in deterministic schedules becoming more expensive and longer than expected, or even becoming unfeasible. Many researches in previous years have been dedicated to solving the stochastic scheduling problem. They have analysed different non-structural (parametric) sources of uncertainty, such as the examination of renewable resource availability and resource consumption by He and Wan [12] and the cost of activity by Chen and Zhang [16] and Xu and Feng [11]. A stochastic activity duration analysis was applied by all the authors [11]–[16].

There are three groups of methods for solving the stochastic scheduling problem: predictive, proactive, and reactive methods [17]. Predictive methods ignore uncertainty, so the predictive schedules can be late, over the budget, or even become infeasible. Proactive methods are intended to construct a predictive schedule that will perform well under a wide variety of external situations. Reactive methods are intended for online scheduling at the time of job execution, incorporating up-to-date information, and changing the schedule when disruptions take place [22].

Proactive methods are the most popular in researches on the stochastic scheduling problem. The main idea of proactive methods is to distinguish the two decision searching stages: the stage of the uncertainty removal and the stage of the deterministic problem solving. The direct order of the stages is used in most of the researches. The following techniques of uncertainty removal are considered by different authors: two-stage algorithm based on chance-constrained programming by He and Wan [12], 99-methods by Zhang and Chen [13], heuristic algorithm with forbidden sets and forward-backward list scheduling by Csebfalvi [14], and Monte-Carlo simulation by Chen and Zhang [16]. The reverse order of the stages was used by Artigues et al. [15]. In this research, first, the initial schedule is found by solving the deterministic equivalent of the stochastic problem obtained by replacing the uncertain parameters with their average values. Second, the schedule is modified by inserting buffer times into the schedule to discourage the propagation of schedule disruptions.

The reviewed studies do not consider the structural uncertainty associated with the insertion of new projects into the schedule, while this problem is real in small and medium-sized enterprises. The current article considers a new proactive scheduling method under structural uncertainty with the use of a genetic algorithm, simulation, and numerical methods.

III. Deterministic Problem Statement

Let us consider the deterministic problem of project scheduling aimed at the calendar planning of operations. All project operations have to be carried out in combination with a set of time constraints. The set of time constraints is defined through negotiations with customers. In the case of the organisation's own lack of resources, subcontracted resources have to be involved to meet the time constraints.

The objective functions of the considered problem are: 1) subcontract cost minimization and 2) minimization of total downtime of own resources. The second objective function is associated with the fixed labour costs in the project companies. If the salaries are fixed then downtime is also paid, which is not profitable for the company.

For the project scheduling problem considered in this study, the following assumptions have been made:

1. A single project consists of a number of operations with a known processing time, early and late start dates, labour input, and labour cost.
2. The operation requires the availability of renewable resources (own or subcontracted workforces).
3. Nonrenewable resources are not considered.
4. Operations cannot be interrupted.
5. Subcontractors can be involved in performing part of the operation.
6. Subcontractors can be interrupted and the operation can continue with the use of the company's own resources in the event of the reappearance of its own available resources.
7. Subcontractors are available every day on request in unlimited quantities.

Let us describe the problem of project portfolio scheduling with the use of the following designations.

Indices:

i: project index, $i = 1, 2, \ldots, P$.
j: operation index, $j = 1, 2, \ldots, Op_i$.
w: department index, $w = 1, 2, \ldots, V$.
t: time index, $t = 0, 1, 2, \ldots, T$.

Decision variables:

$TB(i,j)$: set of start dates of operations.

Initial parameters:

$ES(i,j)$: early start date of the operation (i,j).
$LS(i,j)$: last start date of the operation (i,j).
SL_w: number of persons in the department w.
$SLO(i,j,w)$: amount of workforce (persons) needed in department w to fulfil the operation (i,j).
$SS(i,j)$: operation (i,j) subcontracting cost per day.

Parameters obtained in the decision-making process:

$Active(i,j,t)$: a sign of the operation (i,j) execution at time t.

$$Active(i,j,t) = \begin{cases} 1, & \text{if operation } (i,j) \text{ is executed at the moment } t \\ 0, & \text{otherwise} \end{cases}$$

$RD(t,w)$: resources from department w demanded to fulfil the active operations at the time t.

$$RD(t,w) = \sum_{i=1}^{P} \sum_{j=1}^{Op_i} [Active(i,j,t) \cdot SLO(i,j,w)]$$

$VF(t,w)$: amount of free workforce of department w at the time t.

$$VF(t,w) = \begin{cases} SL_w - RD(t,w), & \text{if } RD(t,w) \le SL_w \\ 0, & \text{otherwise} \end{cases}$$

$V_{SC}(i,j)$: volume of subcontracted workforces on operation (i,j).

Problem description:

$$OF_1 = \sum_{i=1}^{P} \sum_{j=1}^{Opi} (SS(i,j) \cdot V_{SC}(i,j)) \to \min, \quad (1)$$

$$OF_2 = \frac{\sum_{t=0}^{T} \sum_{w=1}^{V} VF(t,w)}{T \cdot V} \to \min, \quad (2)$$

$$TB(i,j) \in [ES(i,j); LS(i,j)] \quad \forall i, \forall j \quad (3)$$

Objective function (1) minimises the total subcontracting cost. Objective function (2) minimises the total downtime of own resources. Constraint (3) maintains the time frame of the operations' start.

IV. Genetic Algorithm Based on Annealing Simulation and Novelty Search

The genetic algorithm (GA) is one of the evolutionary approaches that can be used to solve complex system management problems in a short time [18]. The technique of the GA application includes the following steps: 1) selecting the method of encoding the problem decision (phenotype) into a chromosome (genotype); 2) definition of the evaluation method of the chromosome fitness function (FF); 3) the genetic operator's description; and 4) the initial population generation and GA work. The modification of the GA on the basis of an annealing simulation and novelty search is considered in the article in order to enhance the quality of the decisions on the scheduling problem.

A. Chromosome Encoding

There are various techniques for decision encoding presented in the literature: operations sequence encoding [3][6][10], operations precedence encoding [2][4], operations start dates encoding [5][7], and encoding of resource assignment for the operation [7]–[9]. We used the encoding of the shifting of the operation start dates because this technique supports time constraints, is not redundant, and is simple to implement.

The GA chromosome encodes the operations' start dates, shifting from the initial work plan to the right or left on the

time axis via binary code (0/1). The shift range is two weeks on either side of the initial operation start date. The chromosome size is $5 \cdot r$ genes, where r is the number of analysed operations, 5 is the number of genes needed to encode a single operation shifting (4 genes to encode $2^4 = 16$ shifting days and 1 gene to encode the shifting direction).

B. Genetic Algorithm Modification

The concept of a novelty is a major GA concept. This concept is connected with the emergence of new elements and interactions in the environment during evolution. Two novelty types are distinguished in [23]: 1) *combinatorial novelty*, when the new species emerge by combining the existing species; and 2) *creative novelty*, when the new species are not reproducible by a combination of the species. The validity of the fundamental feasibility of the second novelty type is still open.

Let us consider the case of a combinatorial novelty search as an adaptation mean in an open system. To implement this approach we modify a simple GA by introducing the concept of "decision originality" as a measure of the decision fitness to the environmental conditions [23]. The decision-chromosome's originality in the population is determined via the numerical transformation of the Hamming distance matrix.

Let us define the Hamming distance matrix as follows:

$$H = \left(h_{ij}\right)_{i=1,\,j=1}^{N}, \tag{4}$$

where h_{ij} is the Hamming distance between the i-th and j-th chromosomes (Ch_i and Ch_i), equal to the number of positions, at which the corresponding gene values are different in chromosomes Ch_i and Ch_j; N is the number of chromosomes.

We associate the matrix H with the matrix of originality weights W defined as follows:

$$W = \left(w_{ij}\right)_{i=1,\,j=1}^{N}, \tag{5}$$

where w_{ii} is the weight of the corresponding value of the Hamming distance determined as a quadratic function, increasing in the range from 1 to R as element h_{ij} is changed in the range of 0 to L:

$$w_{ij} = \frac{R-1}{\sqrt{L}} \cdot \sqrt{h_{ij}} + 1, \tag{6}$$

where L is the chromosome size, and R is the maximum weight of the chromosome in the pair, $R > 0$.

The two strategies of chromosome crossing have been described using the concept of originality. The first strategy – *the originality search strategy (OSS)* [24] – focuses on the combinatorial search for the new decisions in the population by crossing chromosomes that have different encodings. The second strategy – *the maximum search strategy (MSS)* [18] – focuses on the targeted search for the best chromosomes by crossing chromosomes that are the most adapted to the environment. The fitness of the i-th chromosome to the environment is evaluated by the fitness function FF_i, $i=1\ldots N$.

Let us define the chromosome crossing probability matrices on the basis of the proposed strategies as follows:

$$P_{OSS} = \left(p_{ij}^{OSS}\right)_{i=1,\,j=1}^{N}, \; p_{ij}^{OSS} = \frac{w_{ij}}{\sum_{j=1}^{N} w_{ij}}, \tag{7}$$

$$P_{MSS} = \left(p_{i}^{MSS}\right)_{i=1}^{N}, \; p_{i}^{MSS} = \frac{FF_i}{\sum_{i=1}^{N} FF_i}. \tag{8}$$

In formulas (7) and (8) the matrices' cells are filled by probability values in accordance with the roulette law [18]. In the case of the OSS strategy, the weight of the chromosome originality serves as a measure of chromosome importance. In the case of the MSS strategy, the chromosome FF serves as a measure of chromosome importance.

An annealing simulation algorithm (ASA) [25] is intended to implement the proposed chromosome crossing strategies during the GA work. This algorithm is based on the analogy of the metal annealing process, which results in the appearance of new metal properties. The technique for ASA and GA integration is proposed below.

Step 1. Set the annealing simulation algorithm parameters: the initial value of the parameter t_Z; the value of the parameter α, which controls the rate of annealing temperature decrease, $0 \le \alpha \le 1$.

Step 2. Set the GA parameters: the number of generations K; the chromosome size L; the likelihood of the genetic operators being applied. Set the number of the current population Z: $Z = 1$. Generate the initial population.

Step 3. Apply the genetic operators to the current population Z with a probability that depends on the value of parameter t_Z. Increase the number of the current population $Z = Z + 1$. Change the value of parameter t_Z [25]:

$$t_{Z+1} = t_Z + \alpha \cdot t_Z. \tag{9}$$

Step 4. Check the condition: $Z > K$. If the condition is satisfied then go to Step 5, otherwise return to Step 3.

Step 5. Stop.

The probability of the genetic operator's application is defined on the basis of the annealing simulation in order to reflect the operator's dynamic nature.

C. Crossover Operator

The probability of selecting the first and second parents from the current population Z for the crossover operator (CO) is described below. The probability of selecting the first parent has to take into account both random selection and targeted selection based on the MSS strategy (8). The probability of random selection should be reduced in the

population's evolution, and the probability of the MSS strategy should be increased. This fact is reflected in the probability of selecting the first parent i in the population *Z*:

$$P_i^Z(CO) = \frac{1}{N} \cdot \left(1 - \exp\left(-\frac{1}{t_Z}\right)\right) + p_i^{MSS} \cdot \exp\left(-\frac{1}{t_Z}\right). \quad (10)$$

The probability of selecting the second parent has to take into account the OSS and MSS strategies. The probability of the OSS applying (7) should be reduced during the population's evolution, and the probability of the MSS applying (8) should be increased. This circumstance is reflected in the probability formula for selecting the second parent *j* for the first parent *i* in population *Z*:

$$P_j^Z(CO) = p_{ij}^{OSS} \cdot \left(1 - \exp\left(-\frac{1}{t_Z}\right)\right) + p_j^{MSS} \cdot \exp\left(-\frac{1}{t_Z}\right). \quad (11)$$

D. Mutation and Inversion Operators

The applied probability of the mutation operator (MO) in population *Z* is described below. This formula has to take into account the probability reducing during evolution in order to save genetic material [24]:

$$P_Z(MO) = P_0(MO) \cdot \left(1 - \exp\left(-\frac{1}{t_Z}\right)\right), \quad (12)$$

where $P_0(MO)$ is the initial value of the mutation operator applied probability.

The applied probability of the inversion operator in population *Z* is described by analogy with the mutation operator applied probability.

E. Fitness Function

The following fitness function considers both objective functions (1) and (2) described in Section III:

$$FF = \omega_1 \cdot (OF_1^{Init} / OF_1) + \omega_2 \cdot (OF_2^{Init} / OF_2) \to \max, \quad (13)$$

where ω_1, ω_2 are weight coefficients, $\omega_1+\omega_2=1$; OF_1^{Init}, OF_2^{Init} are objective function initial values obtained by expert evaluation of the operation start date.

Used FF is described with the use of the linear convolution of normalised heterogeneous criteria (1) and (2).

V. THE DYNAMIC MODEL OF MULTIAGENT RESOURCE CONVERSION PROCESSES (MRCP)

The processes of the project's work execution have to be formalised via a simulation model in order to evaluate the objective function values. The use of multiagent approach at the stage of business process model formalisation is caused by the presence of decision makers in the system; their behaviour is motivated, they cooperate with each other, and accumulate knowledge of the problem domain. The problem of the model selection for business process formalisation was addressed by the authors in [26]. The following models for supporting agent representation of business processes were considered: Gaia model, Bugaichenko's model, Masloboev's model, simulation model of intelligent agents interaction (SMIAI), resource-activity-operation model (RAO), and multiagent resource conversion processes model (MRCP). The comparison showed that the MRCP model has the fullest functionality in area of business process formalisation: it includes a hybrid agent model (intelligent and reactive), a model of a resource converter and a queue system, allowing the analyst to examine the dynamic features of processes [26].

Let us consider the basic principles of the MRCP model.

The MRCP model [27] was developed on the basis of resource conversion process (RCP) model [28] and it targets the modelling of business processes and decision support for management and control processes. A key concept of the RCP model is a resource converter consisting of input, launch condition, conversion module, control block, and output.

The launch condition, once it becomes true, enables the conversion process to take place based on the state of input resources, control commands, available conversion tools, and other external environment events. Conversion time becomes known right before the start of the conversion process as a function of the control commands and active resources limitation.

The MRCP model may be considered as an extension of the basic RCP model, adding the functionality of agents. The main objects of the discrete multiagent RCP are: operations (*Op*), resources (*Res*), control commands (*U*), conversion devices (*Mech*), processes (*PR*), sources (*Sender*) and resource receivers (*Receiver*), junctions (*Junction*), parameters (*P*), agents (*Agent*), and coalitions (*C*). Process parameters are set by the object characteristics function. Relations between the resources and conversion device are set by the link object (*Relation*). The agents and coalitions' existence assumes availability of the situations (*Situation*) and decisions (action plan) (*Decision*).

The MRCP model has a hierarchical structure, defined by high-level integration system graphs. Agents control the RCP objects. Every agent includes a unique model of a decision maker. The agent (software or hardware entity) is defined as an autonomous artificial being with active and motivated behaviour, capable of interacting with other objects within a given virtual environment. With every system tick the agent performs the following operations [27]: environment (current system state) analysis, state diagnosis, knowledge base access (knowledge base [KB] and database [DB] interaction), and decision-making. Thus, the functions of analysis, situation structuring and abstraction, as well as the control commands generation of the resource conversion process are performed by agents.

Consequently, coalition is generated after the union of several agents. Figure 1 shows an example of C_1 coalition formation after the union of A_2 and A_3 agents.

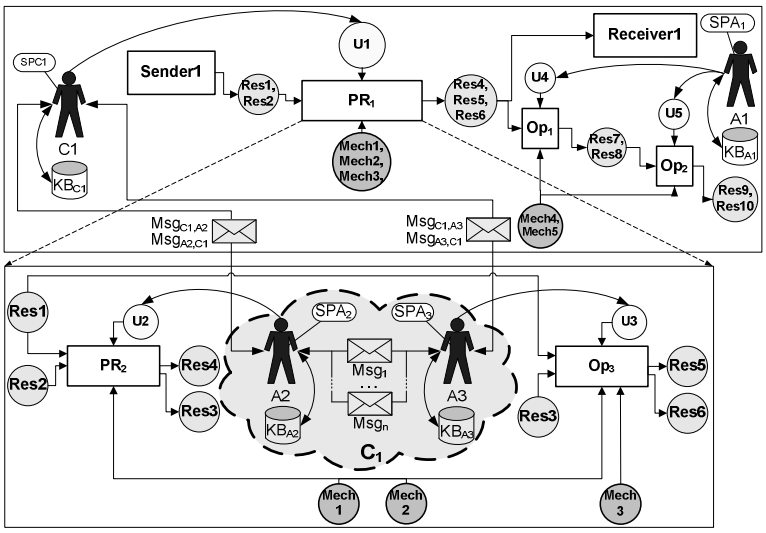


Figure 1. Interaction and coalition formation.

Agent coalition has the following structure:

$C = \langle Name, \{A_1,\ldots,A_m\}, G_C, KB_C, M\_In, M\_Out, SPC, Control\_O \rangle$,

where *Name* is coalition name; $\{A_1,\ldots, A_m\}$ is a collection of agents forming a coalition; G_C is coalition goal; KB_C is coalition knowledge base; *M_In* is a collection of incoming messages; *M_Out* is a collection of outgoing messages; *SPC* is a collection of behaviour scenarios acceptable within coalition; *Control_O* is a collection of controlled objects of the resource conversion process.

The simulation algorithm of the agent-containing model comprises the following main stages: system time determination, agent and coalition actions processing (state diagnosis, control commands generation), conversion rules queue generation, conversion rules execution, and operation memory state (i.e., resources and mechanism values) modification. The simulator makes use of the expert system unit for diagnosis of situations and generation of control commands.

Each agent possesses its knowledge base, set of goals needed for the behaviour configuration setting, and priority that defines agent order in control gaining queue.

The following agent behaviour rules structure was used in the resource conversion processes subject area:

Name <*Rule Name*>

If <*Message Conditions*, *RCP Conditions*, *G_Ag Conditions*>

Then <*G_Ag Changes*, *Message Actions*, *Private Actions*>,

where *Message Conditions* are message-related conditions; *RCP Conditions* are resource conversion process-related conditions; *G_Ag Conditions* are goal-related conditions; *G_Ag Changes* are agent current goals modifying actions; *Message Actions* are message generation actions; *Private Actions* are converters and resource-related actions (activity plan), targeting the achievement of set goals.

The MRCP agent behaviour rules have been developed on the basis of the special-purpose object-oriented Reticular Agent Definition Language (RADL) in the form of **When-If-Then** [29].

Generally, in the case of any situation corresponding to the agent's activity, the agent tries to find a decision (action scenario) in the knowledge base or work it out itself according to the existing behaviour rules, makes a decision, controls goals' achievement, delegates the goals to its own or another agent's RCP objects, and exchanges messages with others.

The MRCP model was chosen to evaluate the chromosome FF value (13). The decision variables and input parameters described in Section III are fed in the model input. The parameters obtained in the decision-making process are the model output. In the MRCP model, we use agents to implement the resource allocation algorithm and use simulation to perform the operation's execution. The resource allocation algorithm is described in [23] and allows executors of operations to be appointed in accordance with the assumptions made in Section III.

VI. MULTIAGENT GENETIC OPTIMISATION PROGRAM

One of the software development problems when addressing the implementation of the proposed scheduling method is the choice of modelling tool. The modelling tool should support the RCP, multiagent, and expert models' description and have built-in, object-oriented development tools in order the additional tool functions to be developed by the systems analyst (programmer).

A. Comparative Analysis of Modelling Systems

Let us consider the following modelling systems: simulation modelling tools PlantSimulation (P) [30] and Simio (S) [31], particularly real-time expert system G2 (G) [32], multiagent simulation systems AnyLogic (A) [33], RepastJ (R) [34], MagentA (M) [35], and BPsim (B) [26]. The results of the comparative analysis of these tools are presented in Table I.

TABLE I. ANALYSIS OF THE MODELLING TOOLS

Comparison criteria	P	S	G	A	R	M	B
RCP modelling availability							
Subject area conceptual model design	○	○	○	○	○	●	●
RCP description language	●	●	●	●	●	●	●
Hierarchical process model	●	●	●	●	○	○	●
Use of natural language for model definition	○	○	●	○	○	○	○
Multiagent modelling availability							
"Agent" element	○	●	○	●	●	●	●
Agents behaviour models	○	●	○	●	●	●	●
Agent's knowledge base support	○	○	○	○	○	●	●
Message exchange language	○	○	○	●	●	●	●
Other modelling techniques availability							
Simulation modelling	○	○	●	●	○	○	●
Expert modelling	○	○	●	○	○	○	●
Situational modelling	○	○	●	○	○	○	●
Evolution modelling	○	○	○	○	○	○	○
Object-oriented approach							
Use of UML language	○	○	○	○	●	○	●
Object-oriented programming	●	●	●	●	●	○	●
Wizard technology for agent design	○	○	○	○	○	○	●
Object-oriented simulation	○	○	●	●	○	○	●
Subject area conceptual model and object-oriented simulation integration	○	○	○	○	○	○	●

As we can see, all the current systems lack the support of some features that might be useful for effective simulation. For example, the problem domain conceptual model design and agent-based implementation approach is limited. Also, some systems, e.g., AnyLogic and G2, require users to have several programming skills. So, for a non-programming user there is no system that can offer a convenient description of a multiagent resource conversion process. Again, AnyLogic and G2 make use of high-level programming language, which results in these products being highly functional.

Not all of the systems support the evolutionary modelling methods. Only the BPsim system includes wizard technology for agent design and tools for the integration of the subject area conceptual model and object-oriented simulation. As a result of the comparison of modelling systems, the BPsim system was selected as the basis for the proposed scheduling method implementation because the system supports the development and integration of the intelligent agents (wizards) with the object-oriented simulation using a common database.

B. BPsim Agent Architecture

The BPsim system consists of the following subsystems: BPsim.MAS dynamic situations modelling system and BPsim.MSN technical and economic development system [26]. BPsim.MAS supports the MRCP model (including reactive agents) description via graphical notation of the resource conversion processes. BPsim.MSN ensures the development of the decision search information technology (intelligent agent) based on the UML sequence diagrams [36] and Transact-SQL database management language [37].

So, the BPsim agent may have a hybrid nature, and it contains two components:

- Intelligent (agent is described via a decision search diagram defined on the UML sequence diagram).
- Reactive (agent is described via production rules and/or frame-based expert system).

Two main agent architecture classes are distinguished [38]:

1. Deliberative agent architecture based on artificial intelligence principles and methods, i.e., knowledge-based systems.

2. Reactive architecture based on a system reaction to external environment events.

The currently existing architectures cannot be defined as purely behavioural or purely knowledge-based, and any designed architecture is hybrid, offering features of both types.

Multiagent resource conversion process architecture, which is implemented in the BPsim, is based on the InteRRaP architecture [39], as the most appropriate for the problem domain.

In accordance with InteRRaP architecture's common concept, the BPsim agent model is represented in four levels (Figure 2).

1. The subsystem of cooperation with other agents corresponds to the following MRCP elements: converters, resources, tools, parameters, and goals. The subsystem of cooperation performs the following actions: generates tasks, transfers messages between agents, processes agent commands (performs resource conversion), and alters the current state of the external environment (transfers situation S_n into state S_{n+1}).

2. The external environment interface and reactive behaviour components are implemented in the form of an agent rules base and inference machine (simulation algorithm).

3. The reactive subsystem performs the following actions: receives tasks from the external environment, places tasks in a goal stack, collates the goal stack in accordance with the adopted goal ranging strategy, selects a top goal from the stack, and searches in the knowledge base. If the appropriate rule is located, the subsystem transfers control to the corresponding resource converter from the external environment. Otherwise, the subsystem queries the local planning subsystem.

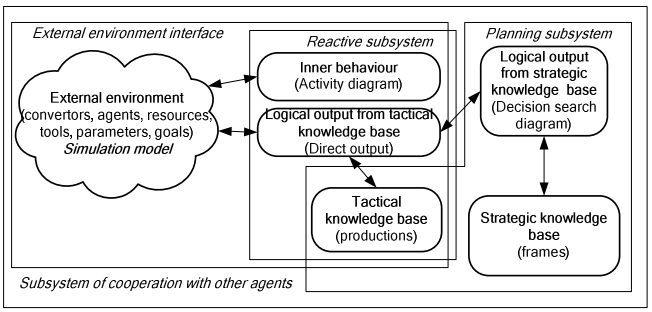


Figure 2. BPsim agent hybrid architecture.

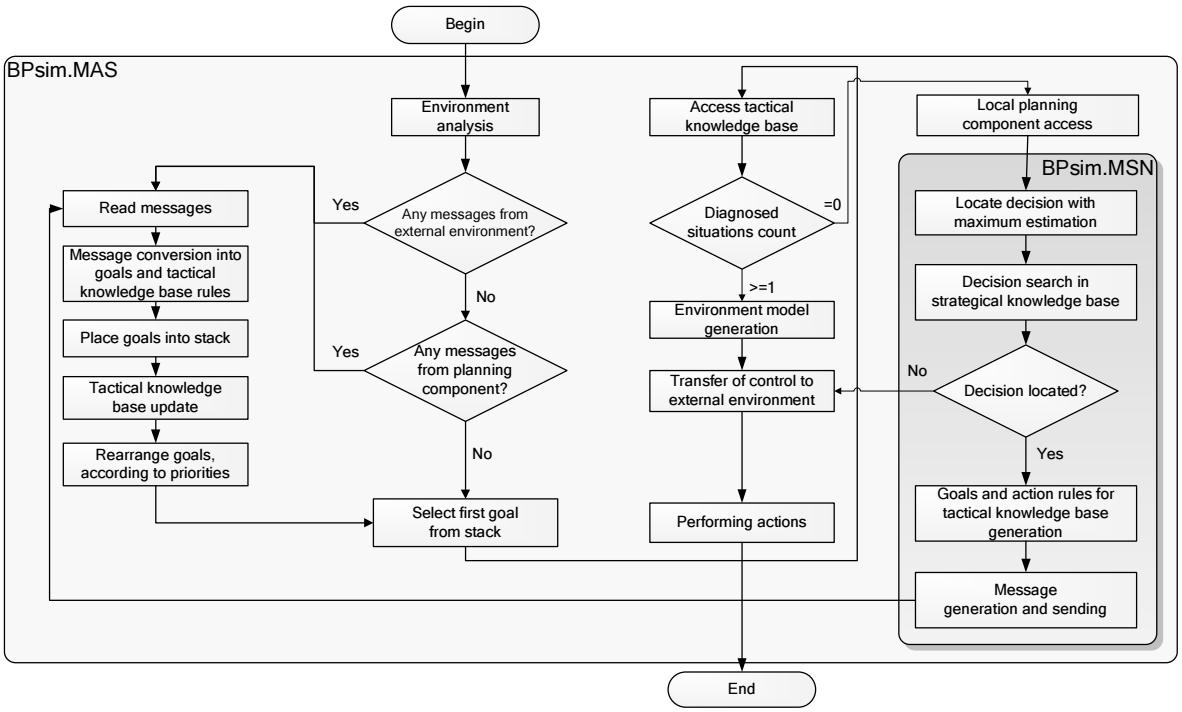


Figure 3. Hybrid BPsim agent activity algorithm.

4. The local planning subsystem's purpose is to search effectively for decisions in complex situations (e.g., when goal achievement requires several steps or several ways of goal achievement are available). The local planning component is built on a frame-based expert system. The frame concept and conceptual graph-based approach are utilised for knowledge formalisation.

5. The scheme presented on Figure 3 shows the interaction of separate units during agent activity within BPsim.MAS and BPsim.MSN.

The problem domain conceptual model and agent knowledge base design are based on the UML class diagram extension. Semantically, this notion may be interpreted as the definition of the full decision search graph, containing all the available ways of goal achievement (pre-defined by experts). The current knowledge base inference machine is implemented in the decision search diagram, based on the UML sequence diagram. Each decision represents an agent activity plan. Each plan consists of a set of rules from a reactive knowledge base. Based on the located decision, the current agent plan is updated. Examination of all available options contained in the knowledge base generates an agent plans library.

If an agent, when processing a task or message received from the external environment, is unable to locate the appropriate rule in its knowledge base (e.g., select an option from several ones), the reactive behaviour component queries the plans library, indicating goal (i.e., task to execute, or external environment state to bring into). The planning subsystem searches the plans library, selects an appropriate plan, and places the first rule of the selected plan into the reactive goals stack.

The problem of the implementation of the BPsim.MAS and BPsim.MSN systems integration is solved by implementing the communication between BPsim agents using a single database.

The communication between BPsim agents is implemented in different ways on the different levels:

1. The message exchange between reactive agents within the dynamic model MRCP is implemented by the transacts' (messages) introduction into the process model and by the description of the message processing rules in the agent's model.

2. The message exchange between reactive and intelligent agents within the dynamic model (MRCP) is implemented via applying the clipboard messages containing common variables used in BPsim.MAS and BPsim.MSN systems.

3. The message exchange between BPsim agents and external systems (in cases when the interaction is necessary to transmit not only data but also knowledge) is implemented by applying the communication protocols. As an interaction standard, the Foundation for Intelligent Physical Agents (FIPA) standard has been selected because it has the following advantages: highest reliability, ontology description availability, problem area compliance, and easy implementation. The Agent Communication Language (ACL) message type, supported in the FIPA standard [40], is used in the message exchange between BPsim agents and the environment.

C. Multiagent Genetic Optimisation Program Development

The multiagent genetic optimisation (MGO) program has been developed on the basis of the BPsim.MAS and BPsim.MSN systems. The MGO program is intended to solve the problem of simulation and evolution modelling integration. The genetic optimisation information technology (IT) has been designed on the basis of BPsim.MSN, and is intended to aid GA setting and GA execution.

The algorithm for the interaction between the decision maker and MGO program during the decision-making process is shown in Figure 4. The MRCP model is intended to conduct the chromosome's FF evaluation by carrying out an experiment with the model. The decoded chromosome phenotype (operations calendar planning) is fed into the model input. The FF evaluation in accordance with (13) is obtained in the model output. Agents in the MRCP model are used to allocate the renewable resources (both own and subcontracted). The decision maker carries out the problem statement and solution choice among the solutions obtained by the use of the MGO program.

The MGO program has a number of advantages (key strengths) compared to existing evolutionary scheduling optimisation software [3][5][9][10]:

1. The integration of simulation, expert, multiagent, conceptual, and evolutionary approaches in order to decide the scheduling optimisation problem.

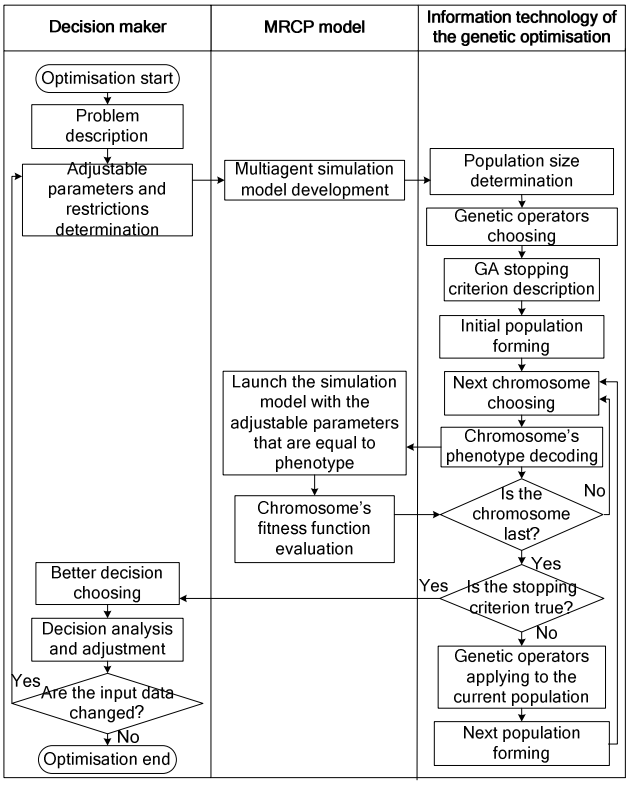


Figure 4. Interaction between the decision maker and MGO program.

2. The availability of the hybrid multiagent system architecture, which allows complex scheduling models to be built consisting of two interacting elements: 1) the dynamic model MRCP and 2) the genetic optimisation model intended for control of model MRCP parameters.

3. Description of the system models using MRCP and UML graphical notation.

4. The evolutionary and simulation models' integration via wizard technology for users without programming skills.

5. The modified genetic algorithm implementation in the genetic optimisation model.

6. Support for the development of the user's own ways of solving the scheduling problem by the use of a modified GA through the decision's phenotype encoding description using Transact SQL query language and decision search diagrams based on the UML (only for user-programmers).

The MGO program is intended to decide the scheduling problem under certainty. Let us consider the modification of the MGO method in order to solve the scheduling problem under uncertainty.

VII. The MGOU Method of Project Scheduling under Uncertainty

The project scheduling process is a time-consuming task that is complicated by the incompleteness of the initial information, the reduction of decision-making time, and increased requirements for the experience and expertise of decision makers (DM). The incompleteness of initial information is related to the uncertainty of the situation in which the decision should operate.

Two different types of uncertainty are allocated in [19]: the uncertainty of the environment state and the "active partner" uncertainty, reflecting the behaviour of the other decision makers. Accounting for the "active partner" uncertainty leads to the problem statement in conflict situations; methods for solving such problems are considered by the theory of games [19].

A. Problem Statement Under Uncertainty

We considered the scheduling problem under the environment's behaviour uncertainty. We associated the environment's behaviour uncertainty in the project scheduling with the lack of information on the number and size of additional projects that may arise during the planning period. The statement of the problem given in Section III takes the following form:

$$y = \{OF_1(\varphi(x,z)), OF_2(\varphi(x,z))\} \to \max,\ x \in X, z \in Z, y \in Y \quad (14)$$

where $X = \{x_1,\ldots,x_n\}$ is a set of the alternative schedules; $Z=\{z_1,\ldots,z_S\}$ is a set of the environment states; $\varphi(x,z)$ is a transformation function of the alternative schedule x at the environment state z in some results; OF_1,OF_2 are objective functions of the problem considered; $Y = \{y_{ij}\}_{i=1..n}^{j=1..s}$ is a set of the decisions evaluation (matrix of decisions).

Each element of the matrix (14) y_{ij} is a collection of 2 estimates of the alternative x_i outcome for environmental condition z_j.

We denoted the set of the probabilities that the environment will be in the states Z as follows: $P= \{p_1,\ldots,p_S\}$, $\Sigma p_j = 1$. The vector P can be unknown for the selected environmental states Z.

The main difficulty in solving the problem (14) is the function φ definition, since this function should consider the cumulative system behaviour statistics, dynamic system processes, and DM behaviour scenario that it is not always possible to represent analytically for real management problems. The use of multiagent simulation allows the imposed requirements of the transformation function φ to be taken into account. The MRCP model is used for the system processes' formalisation and representation of the function $\varphi(x,z)$.

There are various methods of multi-criteria decision making that reduce the multi-criteria problem to the single-criterion problem [19].

In the main criterion method, one of the functions is selected as the objective function. This function best reflects the purpose of the decision making from the user's point of view. Such a transition is not always equivalent to the original problem.

The linear convolution method allows the criterion vector to be replaced by a scalar via a linear combination of all the weighted criteria functions. It is a requirement that all the functions' values should be presented on a numerical scale.

The maximin convolution method is focused on the worst case and chooses the optimality criterion, which corresponds to the smallest value of all the criteria. The disadvantage of this method is its focus on the worst criteria.

In the presence of knowledge about the decision maker preferences, the criteria coefficients can be determined using the Saaty method [41]. The main object of the Saaty method is a triangular matrix S of pairwise comparisons, each element of which is interpreted as the superiority ratio of the one criterion over the other. Coefficients of superiority can be selected by the user from a fixed Saaty scale. Further, the matrix S is transformed to the new matrix S', for which the maximum eigenvalue is determined through a linear equations system. The weighted coefficients vector of the individual criteria significance is an eigenvector of the matrix S' corresponding to the maximum eigenvalue of the matrix S'. The Saaty method's disadvantages are the need to solve linear systems of equations and the quadratic dependence of the pairwise comparison's number on the criteria and number of alternatives [41].

The linear convolution method was selected to reduce the considered multi-criteria scheduling problem to the single-criterion problem using normalisation of the criteria with respect to the reference values (13).

Reduction of the problem under uncertainty (14) to the deterministic problem implies the application of numerical criteria. These criteria reflect the optimistic/pessimistic perspective of the decision maker on the processes [19]. The optimistic criterion is the Hurwitz criterion with the coefficient equal to 1. The pessimistic criteria are: Wald criterion, Savage criterion, and Hurwitz criterion with coefficients equal to 0. The neutral criteria are: Bayes-Laplace criterion, Bernoulli criterion, and Hurwitz criterion with coefficients equal to 0.5.

All the mentioned criteria are used for uncertainty removal in the scheduling method described below.

B. Multiagent Genetic Optimisation Method Under Uncertainty

The multiagent genetic optimisation method under uncertainty (MGOU method) integrates simulation, genetic algorithms, and numerical methods. The MGOU method includes the following steps.

Step 1. Definition of the input information: a) sets Z of the environment state and sets P of the probabilities that the environment is in the certain state; b) the function φ with the use of MRCP model. The alternative work schedule x and environment state z are fed into the model input. The OF_1 and OF_2 evaluations are obtained in the model output.

Step 2. Formation of a set of alternative schedules X, that include the efficient (optimal) solution of the problem (1–3) under conditions of certainty. At this step the MGO method is used to find the chromosome population (set of alternative work schedules) including the efficient (optimal) solution of the problem (1–3).

Step 3. Formation of the matrix of decisions Y for the problem (14) via conducting $n \cdot s$ experiments with the MRCP model with n alternatives from the set X and s alternatives from the set Z. The matrix of decisions Y for the problem (14) is presented in Table II.

Step 4. Replacement of the optimality criterion vector $\{OF_1,OF_2\}$ of the problem (14) on the scalar value. The replacement can be performed using known numerical methods, such as the linear convolution as in formula (13). The statement of the problem (14) after the step's fulfilment takes the form:

$$y = F(x,z) \to \max,\ y \in Y, x \in X, z \in Z, \qquad (15)$$

where $F(x,z)$ is the function of implementing an alternative x in environment state z to decision evaluation y.

Step 5. Reduction of the problem under uncertainty (15) to the deterministic problem using numerical functions J. There are several numerical functions, depending on the knowledge of probabilities vector P and the strategy used for the uncertainty removal: the Bayes-Laplace criterion (mathematical expectation criterion), Wald criterion (criterion of the guaranteed result, maximin criterion), Savage criterion (criterion of minimum regret), Bernoulli criterion (principle of insufficient grounds), and Hurwitz criterion (criterion of pessimism-optimism) [19].

TABLE II. MATRIX OF DECISIONS FOR THE MULTI-CRITERIA PROBLEM UNDER UNCERTAINTY

	$\mathbf{z_1}$	…	$\mathbf{z_S}$
$\mathbf{x_1}$	$OF_1(\varphi(x_1,z_1))$, $OF_2(\varphi(x_1,z_1))$	…	$OF_1(\varphi(x_1,z_s))$, $OF_2(\varphi(x_1,z_S))$
…	…	…	…
$\mathbf{x_n}$	$OF_1(\varphi(x_n,z_1))$, $OF_2(\varphi(x_n,z_1))$	…	$OF_1(\varphi(x_n,z_s))$, $OF_2(\varphi(x_n,z_S))$

Let us consider the description of selected numerical functions of uncertainty removal.

The Bayes-Laplace criterion is applied with knowledge of probabilities P and characterises the "average income" when making an alternative work schedule x [19]:

$$J_{BL}(x) = \overline{F(x,z)} = \sum_{i=1}^{S} p_i \cdot F(x,z_i) \to \max_{x \in X}, \quad (16)$$

where the line on top of the symbol denotes the mathematical expectation. The decision of the problem (16) will be an alternative work schedule x_{i*}: $i^* = \arg(\max J_{BL}(x_i))$.

The Wald criterion characterises the best solution for the most unfavourable situation [19]:

$$Jv(x) = \min_{z_i \in Z} F(x,z_i) \to \max_{x \in X} \cdot \quad (17)$$

This formula is valid if the function $F(x,z)$ characterises the "income". Otherwise, the maximin criterion is transformed into a minimax criterion.

The Savage criterion characterises the best solution when comparing the worst losses. The losses emerge when there is preference of others for one alternative x at a fixed environment state z [19]:

$$Js(x) = \max_{z_i \in Z} \left(\max_{x_j \in X} F(x_j,z_i) - x_j \right) \to \min_{x \in X} \cdot \quad (18)$$

The Bernoulli criterion characterises decisions by considering equiprobable external environment events [19]:

$$Jb(x) = \frac{1}{S} \cdot \sum_{i=1}^{S} F(x,z_i) \to \max_{x \in X} \cdot \quad (19)$$

The disadvantage of this criterion is that the unknown distribution law of the magnitude P is replaced by the uniform distribution law.

The Hurwitz criterion characterises the solution for a given propensity DM to pessimism or optimism [19]:

$$Jh(x) = \alpha \cdot \max_{z_i \in Z} F(x,z_i) + (1-\alpha) \cdot \min_{z_i \in Z} F(x,z_i) \to \max_{x \in X}, \quad (20)$$

where α is the indicator of pessimism or optimism. If $\alpha = 0$ the case of extreme pessimism comes about, if $\alpha = 1$ the case of extreme optimism comes about.

VIII. MULTIAGENT GENETIC OPTIMISATION PROGRAM UNDER UNCERTAINTY

The program for multiagent genetic optimisation under uncertainty (MGOU program) has been developed on the basis of an MGO program, BPsim.MAS dynamic situations modelling system and BPsim.MSN technical and economic development system. The information technology for uncertainty removal has been designed on the basis of BPsim.MSN.

The algorithm for the interaction between the decision maker and MGOU program during the decision-making process under uncertainty is shown in Figure 5. The decision maker carries out the problem statement, definition of the different environment condition, and alternative work scheduling set formed with the help of the genetic optimisation IT.

The uncertainty removing IT is intended to aid risk assessment under the uncertainty behaviour of the environment. The MRCP model is intended to conduct the experiments according to the plan for different values of work scheduling and project size (environment condition).

The MGOU program has a number of advantages (key strengths) compared to existing scheduling optimisation under uncertainty software [11]–[13][15]–[17], as well as the advantages of the MGO program that have already been described in the Section VI:

1. The integration of simulation and evolutionary approaches with numerical decision support methods to decide the optimisation problem of scheduling under uncertainty.
2. Consideration and removal of parametric and structural environmental uncertainty.
3. The availability of uncertainty removal via wizard technology for users without programming skills.

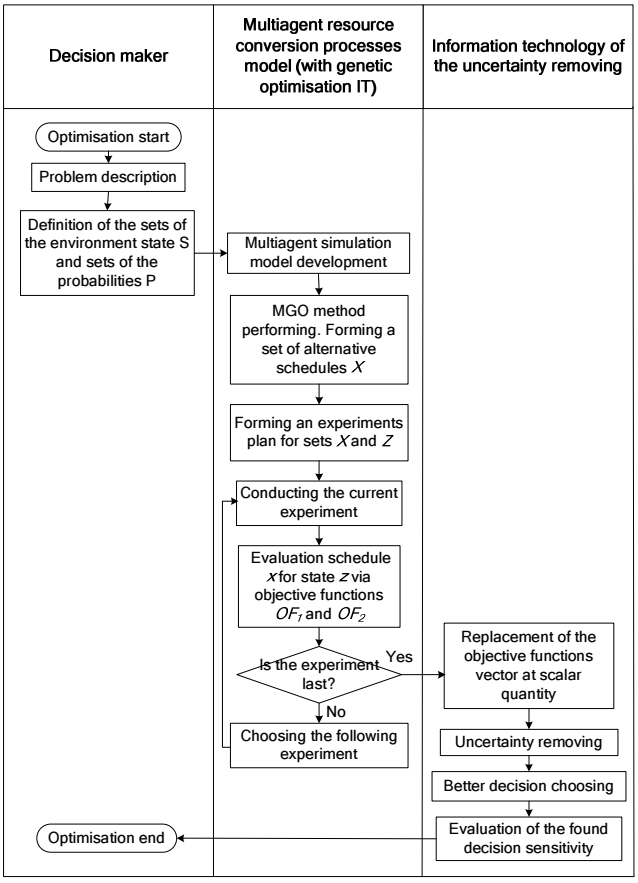


Figure 5. Interaction between the decision maker and MGOU program.

IX. Comparative Analysis of the Project Scheduling Methods

Let us consider the following project scheduling methods: critical path method (CPM) and program evaluation and review technique (PERT), branch and bound method (B&B), genetic algorithms, Xu and Feng's method, Osaba's method, and the MGOU method. A comparison of the selected methods is presented in Table III.

The CPM and PERT methods allow the reserves of time for the execution of certain works to be determined [17][42]. CPM assumes the deterministic duration of activities and PERT incorporates uncertainty into activity durations.

The branch and bound method uses the evaluation of upper and lower bounds to cut a set of solutions via removal of the subsets containing no optimal solutions. The upper bound is obtained using heuristics, while the lower bound can be found using mathematical programming [17][42][43].

Genetic algorithms are used to find the optimal schedule via the evolution of populations of schedules with the help of genetic operators [2]–[10][18]. The optimal solution can be found in GA by considering not only the one improvement decision but many improvement decisions.

Xu and Feng's method is intended to optimise project scheduling with uncertain activity durations and activity costs [11]. First, fuzzy random parameters are transformed into fuzzy variables that are subsequently defuzzied using an expected value operator with an optimistic-pessimistic index. Second, the deterministic problem is solved with the use of a hybrid particle swarm optimisation algorithm.

The Osaba's method is intended to solve the dynamic travelling salesman problem with the use of integrated simulation and genetic algorithms [10]. Simulation is used to reflect the dynamic nature of the system processes and objective function.

TABLE III. Analysis of the scheduling methods

Comparison criteria	CPM, PERT	B&B	GA	Xu/Feng meth.	Osaba meth.	MGOU meth.
Problem statement						
Scheduling	●	●	●	●	●	●
Renewable resources	●	●	●	●	●	●
Nonrenewable resources	○	○	○	●	○	●
Subcontract optimisation	●	○	○	○	○	●
Methods for solving the deterministic problem						
Simulation	○	○	○	○	●	●
Multiagent modelling	○	○	○	○	○	●
Optimisation methods: exact/heuristic	●/○	●/○	○/●	○/●	○/●	○/●
Uncertainty consideration						
Parametric uncertainty with: distribution law/fuzzy logic	●/○	○/○	○/○	○/●	○/○	●/○
Structural uncertainty	○	○	○	○	○	●

The MGOU method integrates the simulation, multiagent, and evolutionary modelling and numerical methods in order to solve the project scheduling problem under uncertainty.

The following comparison criteria were distinguished: application of renewable and nonrenewable resources; optimisation of subcontracting volume in order to decrease the project costs; application of simulation in order to adequately formalise the nonlinear, non-convex, and non-differentiable system processes model; application of multiagent model in order to reflect the decision makers model; application of exact and heuristic optimisation methods in order to conduct the optimisation experiment for optimal solution finding; consideration of the uncertainty with different description in order to reflect unexpected external influences on the schedule.

As we can see from the table, all methods except the MGOU method lack the support of some features that might be useful in effective decision searching of the scheduling problem. For example, subcontract optimisation (except CPM and PERT), agent-based approach implementation, and structural uncertainty evaluation are limited. Another disadvantage of the four most popular scheduling methods (CPM, PERT, B&B, GA) are lack of nonrenewable resource consumption (including the resource life cycle description), lack of simulation that helps to analyse the dynamic system processes of the resources allocation, and lack of uncertainty consideration (except PERT). The Xu and Feng's method considers nonrenewable resource consumption and uncertainty with fuzzy logic, but does not use simulation and multiagent models to optimise subcontracted work. The Osaba's method includes simulation but does not consider uncertainty.

The full potential of scheduling under uncertainty is implemented in the MGOU method. The disadvantage of the method is its lack of fuzzy uncertainty description.

X. Experimental Results

The application of the MGO and MGOU programs to solve the project scheduling problem is presented in this section. Let us consider a company, Telesystems, which consists of project, manufacturing, and supply departments. The goal is the minimization of the company departments' total downtime and the total cost of the subcontract. In this section we consider the application of the MGO method to both the project scheduling problem without uncertainty and to the project scheduling problem under uncertainty.

A. Experimental Results for the Deterministic Problem

A detailed statement of the considered problem is given in [21]. The MRCP model has been developed to evaluate the chromosome FF (13). The MRCP model implements the resource allocation model, which satisfies the assumptions determined in Section III. The model adequacy has been proven in [21] through the evaluation of 5 projects. The following input information has been used in the model: 1) number of projects – 10 with 35 operations; 2) time interval – T = 430 days (1 year and 3 months); and 3) time limit – the early and late start of the operations is determined by the

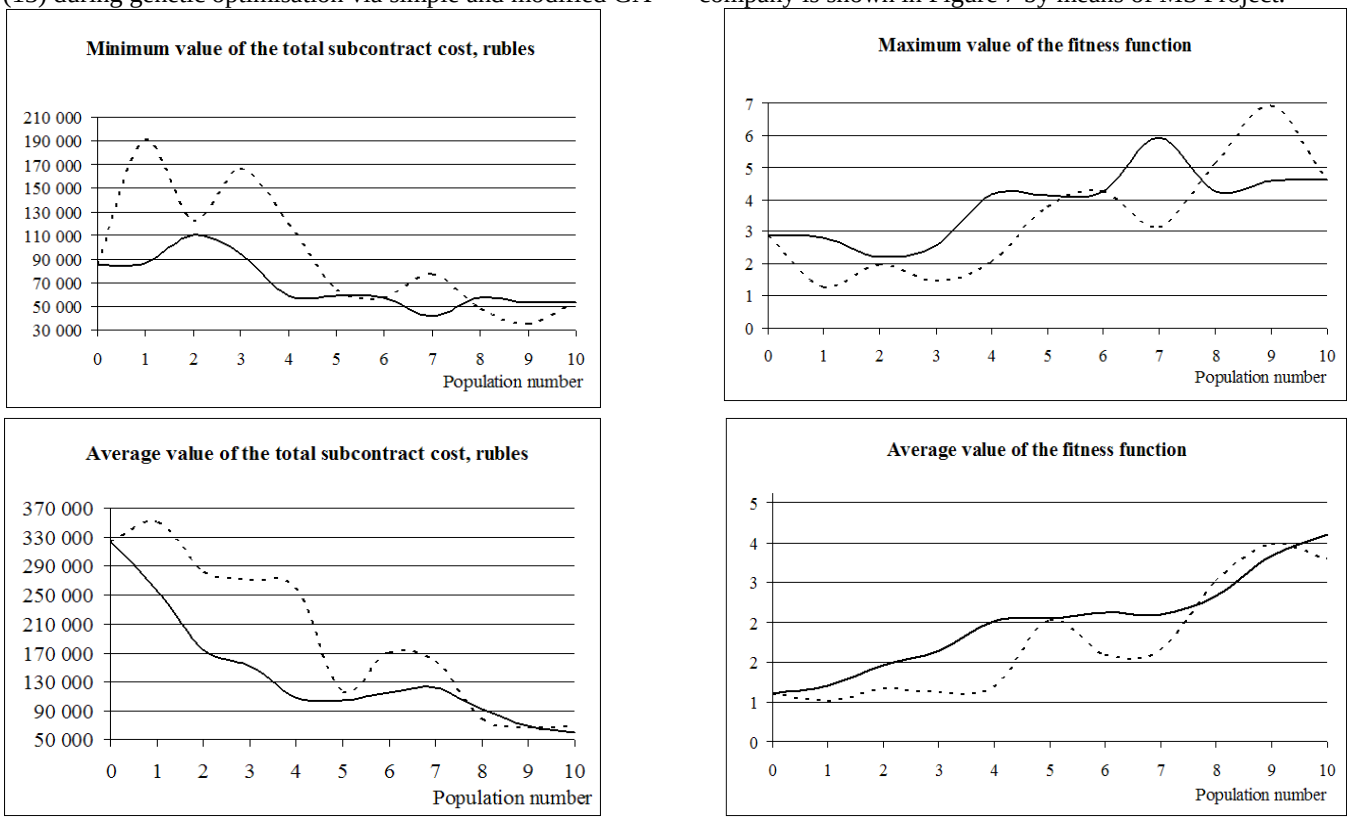
shift in the provisional operational start dates by 2 weeks to the right or left along the time axis.

The following GA parameters have been determined in the course of the genetic optimisation IT work: the population size – 10 chromosomes; the chromosome size – 175 genes (5 genes to encode the 35 project operations); the following genetic operators – reproduction based on roulette, five-point crossover with probabilities determined by (10) and (11), five-point mutation with an initial probability equal to 10% and dynamic probability determined by (12), inversion with initial probability equal to 5%; algorithm stopping criterion – a change of 10 populations; random initial population; and following the ASA parameters values of $t_{Z0} = 1$, $\alpha = 0.9$, $K = 10$.

A comparison of the application of simple GA and modified GA was performed. For a better comparison both algorithms proceeded from one initial population. The dependencies of the chromosome FF and the scheduling problem objective function values from the population number were obtained as a result of genetic optimisation using the developed MGO program. The change in the minimum value of the objective function (1) during genetic optimisation via simple and modified GA is shown in Figure 6.a. The change in the maximum value of the fitness function (13) during genetic optimisation via simple and modified GA is shown in Figure 6.b. The problem solution involves the maximization of the GA fitness function and minimization of subcontract cost.

At the initial stage of the modified GA (change of the generations 1–5) the search of the original decisions predominates and leads to the FF value variations, which does not always ensure the achievement of the best FF values compared with the simple GA. However, the search results are the basis of the targeted search for an extremum in the later stages of GA (change of the generations 6–10) that leads to the higher quality of the solution found. For the problem considered, the decision found with the use of the modified GA leads to the subcontract cost of 35189 rubles, which is 14% below the subcontract cost obtained by using simple GA (41050 rubles). The best decision is achieved in the ninth population.

The project scheduling problem for the Telesystems company has also been solved by use of the MS Project 2007 resources reallocation method and heuristic-simulation (HS) method described in [21]. MS Project 2007 provides the opportunity for resource reallocation (with smoothing) in order to avoid exceeding the own renewable resources availability. The percentage utilisation of the manufacturing department for the initial work plan for the Telesystems company is shown in Figure 7 by means of MS Project.



a. b.

Figure 6. Dependencies of the fitness function and objective function values from the population number when applying simple GA (solid line) and modified GA (dotted line).

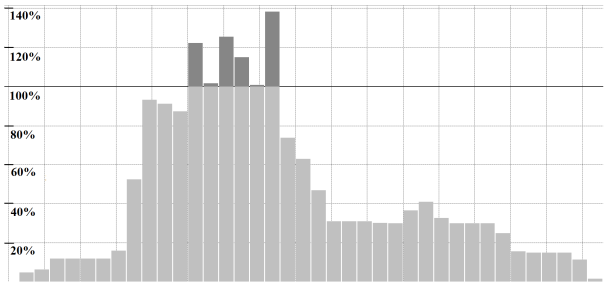
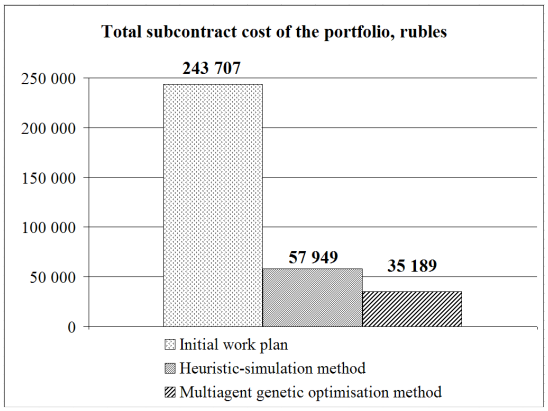


Figure 7. Percentage utilisation of the manufacturing department for initial work plan in MS Project.

The initial work plan has been formed by a decision maker. In the figure, the x-axis shows the time intervals (each of which lasts 12 days); the y-axis shows percentage utilisation. The overallocated resource availability (time intervals where the use of subcontracting is necessary) is shown in the figure by the dark shading of the stripes above the horizontal line at the 100% utilisation level. The application of the MS Project resource reallocation method has allowed the total subcontract cost to be reduced to zero; that is, the objective functions (1) and (2) have reached their optimal values. But the time constraints (3) have not been satisfied by the use of this method. In this way, the MS Project resource reallocation method is not considered suitable for the scheduling problem.

The HS method is based on the analysis of the MRCP model output parameters. In the HS method, the following steps are performed [21]: 1) modelling the results analysis of the subcontract cost and company resources utilisation; 2) search for bottlenecks associated with operations that require high costs of subcontracting; 3) shifting the start dates of operations to the period determined by HS information technology; and 4) transferring the adjusted model at the experiment stage and experiment results evaluation.

Histograms of the objective functions (1) and (2) obtained by the MGO and HS methods are shown in Figures 8.a and 8.b compared with the initial work plan. The total subcontract cost and total downtime of the manufacturing department has been consistently reduced by the use of HS and MGO methods. All time constraints have been satisfied.



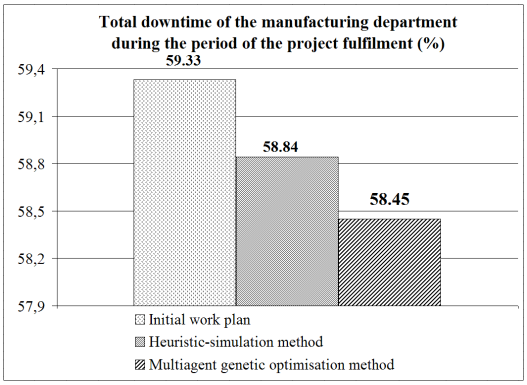
a.

Based on the analysis of the results it was concluded that the MGO method is more effective than the HS method in addressing the project scheduling problem in terms of economic effect. The total subcontract cost of the project portfolio has been reduced by 30% and the total downtime of the manufacturing department has been reduced by 1.5% for a six month period using the MGO method compared to the HS method. The total subcontract cost has been reduced by 7 times using the MGO method compared with the initial work plan. Applying the genetic optimisation based on the simulation and evolutionary modelling integration enhances the efficiency of the decision making by taking into account the dynamic resource allocation model in the simulation model and the fulfilment of the direct search in the decision space by the GA. The economic effect of applying the MGO program to solve the scheduling problem for the Telesystems company will be 430000 rubles per year, which is 9% higher than the economic effect of the use of the HS method to solve the same problem.

Let us compare the HS and MGO methods in terms of performance by measuring CPU time. The CPU time for the HS method T_{HSM} consists of the sum of the HS IT runtime T_{HSIT} and the model MRCP runtime T_{MRCP}. The sum is multiplied by the number of experiments $X_{Iterations}$ conducted during the HS technology work. Thanks to the fact that $T_{HSIT} << T_{MRCP}$ we can neglect the term T_{HSIT} and define T_{HSM} time as follows: $T_{HSM} = X_{Iterations} \cdot T_{MRCP}$.

The CPU time for the MGO method T_{MGO} consists of the sum of the genetic optimisation IT runtime T_{GOIT} and model MRCP runtime T_{MRCP}, which is multiplied by the chromosome number N. The sum is multiplied by the generation number K. Thanks to the fact that $T_{GOIT} << T_{MRCP}$ we can neglect the term T_{GOIT} and define T_{MGO} time as follows: $T_{MGO} = K \cdot N \cdot T_{MRCP}$.

For the real-world scheduling problem the following parameter values were used: $X_{Iterations} = 3$, $K = 10$, $N = 10$. In this case, the HS method is more desirable in terms of performance and consumes 33 times less CPU time than the MGO method. This is connected to the use of the simulation model in the GA for fitness function evaluation, which is performed $K \cdot N$ times. The CPU time of the MGO method is equal to 30 minutes.



b.

Figure 8. Dependencies of the objective function values on the decision-seeking method.

B. Experimental Results for the Problem under Uncertainty

Let us consider the uncertainty behaviour of the environment associated with the appearance of four additional projects in the spring, summer, autumn, and winter, respectively. We defined the sets Z and P through the different events occurring in the system $L = \{l_1,\ldots,l_r\}$ and a set of probabilities of the events' occurrence $P_L=\{p_L(l_1),\ldots,p_L(l_r)\}$; and we considered the set L consisting of the following $r = 8$ events: appearance/absence of additional complex project in each of the seasons.

The graph of the determined environment states Z by using events L is shown in Figure 9.

Sixteen environment states $\{SA,...,SP\}$ were allocated (s=16) as a result of the description of the graph of system states. Each state is characterised by the simultaneous execution of h events from the set L: $z_i \Leftrightarrow \exists L_i^* \subseteq L : L_i^* = l_i^1 \wedge l_i^2 \wedge \ldots \wedge l_i^h$, where $l_i \in L$, $z_i \in Z$, $i = 1..s$, $h \le r$.

The probability p_i of the system's being in the state z_i according to the [19] is determined by

$$p_i = \prod_{j=1}^{h} p_L(l_i^j),\ l_i^j \in L_i^*,\ p_L(l_i^j) \in P_L,\ \sum_{i=1}^{s} p_i = 1. \quad (20)$$

For the problem considered $r = 8$, $h = 4$, $s = 16$; that is, 8 events are considered that define the 16 states of the system, and each state is specified by the simultaneous performance of 4 events. The probabilities of the events' occurrence are specified in Table IV.

The probabilities of the systems being in the environment states z_i calculated for the selected initial conditions are shown in Table V.

Let us calculate, for example, the probability of the systems being in the state $z_{16} = SP$: $L_{16}^* = l_1 \wedge l_3 \wedge l_5 \wedge l_7$ (according to Figure 9). We use the formula (20):

$$p_{16} = p_L(l_1) \cdot p_L(l_3) \cdot p_L(l_5) \cdot p_L(l_7) = 0.75 \cdot 0.5 \cdot 0.5 \cdot 0.25 = 0.05.$$

The model MRCP of the project work performance was used to evaluate the work schedules for 16 selected states of the environment. The MRCP model, which has been described for the MGO method, was supplemented with the following input parameters: marks of the systems being in one of the analysed states.

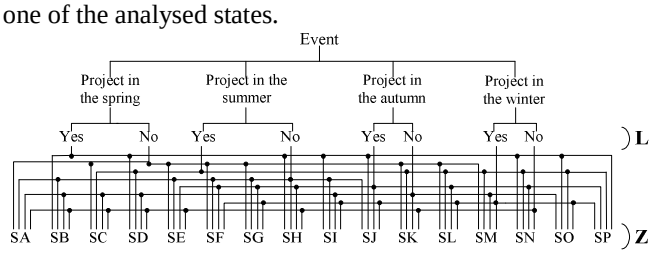


Figure 9. Graph of the system states.

TABLE IV. EVENTS L AND PROBABILITIES P_L OF EVENTS' OCCURRENCE

l_i	Spring project l_1	Summer project l_3	Autumn project l_5	Winter project l_7
$p_L(l_i)$	0.75	0.5	0.5	0.25

As a result of the application of the MGO method, 10 generations of chromosomes were obtained with information stored in the genes about the start dates of the projects' operations. Let us choose as a set of alternatives X the decoded chromosomes in the ninth population of GA, where the best solution to the problem considered was obtained according to Figure 6. Let us define the dimension of the set of alternatives $n = 8$.

The matrix of decisions Y for the project scheduling problem under uncertainty is formed by assessing, with the use of MRCP model, the selected alternatives $\{x_1,\ldots,x_8\}$ via the set of criteria $\{OF_1,OF_2\}$ for each system state $\{z_1,\ldots,z_{16}\}$. Table II has been filled as a result of conducting the $n \cdot s = 128$ experiments. The criterion vector $\{OF_1,OF_2\}$ has replaced the scalar value with the use of the formula (13) and following the values of the formula coefficients: ω_1=0.5; ω_2=0.5. The removal of uncertainty has been carried out by applying the removing uncertainty IT that implements selected numerical criteria.

We applied the Bayes-Laplace criteria (16) for replacement of the matrix of decisions Y on the vector of decisions $J_{BL}(x)$. After performing the transformation, the following vector was obtained: $J_{BL}(x)$ = {104.0; 95.8; 90.4; 121.5; 120.7; 127.7; 143; 126.5}. It is easy to determine that the best solution is an alternative x_7 with the criterion value $J_{BL}(x_7)$ = 143. The data obtained agree with the results of the MGO method application.

Let us investigate the stability of the solution x_7 when changing the initial search conditions. By the term "stability of the solution" we mean the preservation of the solution's advantages over alternative solutions when changing the decision-maker preferences in the evaluation of the importance of objective function (13) criteria and the probabilities of events if they are known. The stability of the found solution x_7 can be evaluated by the application of the Bayes-Laplace criterion to remove the uncertainty of the alternative sets. A series of experiments were carried out in order to find a vector function for evaluating the alternatives X for different initial conditions. The initial conditions were obtained by varying the degree of importance of criteria in forming the implementation function (coefficients ω_1 and ω_2 in formula (13)) and the probabilities of events pL (see Table IV).

TABLE V. ENVIRONMENT STATES Z AND PROBABILITIES P

z_i	SA	SB	SC	SD	SE	SF	SG	SH	SI	SJ	SK	SL	SM	SN	SO	SP
p_i	0.05	0.14	0.05	0.14	0.05	0.01	0.01	0.14	0.05	0.05	0.05	0.01	0.01	0.14	0.05	0.05

The dependence of the Bayes-Laplace function on the changes in initial conditions is shown in Figure 10. Analysis of the behaviour of the function $J_{BL}(x)$ for different initial conditions has shown that the optimal value of the alternative x_7 is preserved for all the analysed situations. Superiority of the calendar plan x_7 is maximum in the case of the equiprobable occurrence of the four additional projects (with a probability of no more than 0.75) and when the objective function OF_1 (which minimises the total subcontracting cost) is selected as the most important criterion of the problem considered. The superiority of the calendar plan x_7 is minimum (right side of the upper chart on Figure 10) in the case of high probabilities of four additional projects.

Let us apply the 4 remaining numerical criteria for removing uncertainty and let us evaluate the stability of the solution x_7 relative to the alternatives set X by changing the coefficient values ω_1 and ω_2 in formula (13). For the Hurwitz criterion we define $\alpha = 0.5$. The dependence of the function $J(x)$ behaviour (for different numerical functions) on the significance of the individual components of the criterion function (13) is shown in Figure 10.

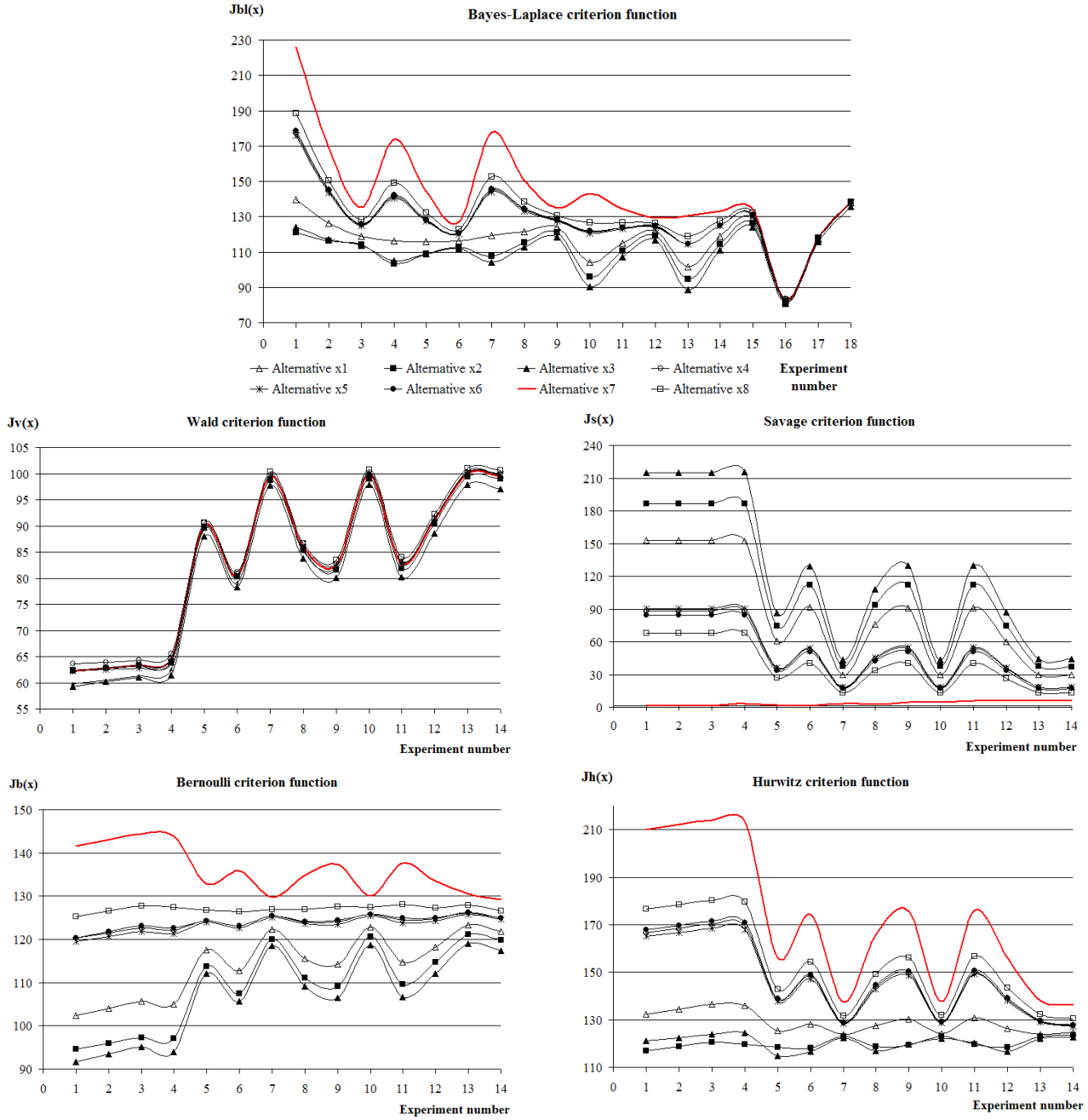


Figure 10. Values of the function J(x) for various numerical criteria when the coefficients of the objective function (13) are changed.

As follows from the charts, the use of the Savage, Bernoulli, and Hurwitz criteria reveals the optimal solution x_7 that maintains stability when changing the function (13) coefficients. The scatter of the function $J(x)$ values is observed for experiments in which function OF_1 is selected as the most important criterion of the problem considered. Function $J(x)$ values approximation is observed for the experiments in which function OF_1 becomes insignificant when compared with the remaining component (function OF_2). In applying the Wald criterion, the correlation between function $J(x)$ values is set for each experiment. This fact is connected only to the analysis of the worst situations in which all alternatives provide comparable outcomes for each experiment.

The solution (calendar plan of the works) that provides the best outcome from the perspective of subcontract cost and the minimization of own resources downtime, and which provides resistance to external factors, was identified with the use of the MGOU method. Also, this solution is optimal for the scheduling problem under certainty. We concluded that the cost of subcontracting is the most important criterion, which greatly affects the objective function when changing environmental conditions.

XI. Conclusion and Future Work

In this paper, a multiagent genetic optimisation method used to solve the deterministic and stochastic project scheduling problem has been described on the basis of the annealing simulation algorithm, novelty search algorithm, genetic algorithm, multiagent simulation, and numerical methods. In order to reflect the dynamic nature of the genetic operators applied, the method combines three different decision-seeking strategies: a random search strategy, originality search strategy, and maximum search strategy. The proposed integration of evolutionary modelling and simulation limits the search space and adequately evaluates the dynamic fitness functions of the chromosomes. The method described has been implemented in MGO and MGOU programs built on the basis of the BPsim.MAS multiagent modelling system and BPsim.MSN development system. The programs integrate simulation, expert, multiagent, conceptual, and evolutionary modelling with numerical methods of uncertainty removal. The comparative analysis of the scheduling methods has shown the disadvantages of the four the most popular scheduling methods (CPM, PERT, B&B, GA) with regard to the lack of nonrenewable resource consumption, simulation, and uncertainty consideration. Also, the analysis has shown the advantages of the MGOU method in the presence of subcontract optimisation, agent-based approach implementation, and structural uncertainty evaluation. The MGO method's application to a real-world deterministic project scheduling problem was compared with the MS Project and HS methods. The MS project resource reallocation method was found to be unsuitable for the scheduling problem under consideration because of the lack of constraints considered. As a result of the comparison between the MGO and HS methods, an improvement in decision quality under the constraints considered has been achieved using the MGO method.

The disadvantage of the MGO method is the high CPU time, which is 33 times higher than that of the HS method. This fact imposes constraints on the GA generation size (no more than 10 chromosomes) and GA iteration number (no more than 10 generations). Different ways of enhancing the applied GA convergence should be considered in future work to meet the described constraints.

The MGOU method of multi-criteria decision making under uncertainty has been applied to the project scheduling problem aimed at optimising the utilisation of subcontracted workforces and own resources. The decisions found by the MGO method have been analysed under the structural uncertainty of the four additional projects appearance. Inferences have been drawn about the optimal decision flexibility with the environmental condition changes. The results of the experiments have shown the coherence of the use of selected numerical criteria.

The aim of future research is to improve the rate of convergence of the proposed genetic algorithm by applying elitism and taboo algorithms. The dependency between the decision search time and problem dimensions is assumed to be established for the MGOU method. Also, consideration of nonrenewable resource allocation and fuzzy description of uncertainty is planned.

It is planned to extend and apply the developed method for the scheduling of technological logistics in the field of metallurgy. Similar problems have the following features: first, the presence of a plurality of industrial units and vehicles to be scheduled and, secondly, the presence of conflict situations when driving vehicles (cranes and steel teeming ladle cars in the shops). The technological logistics scheduling is complicated by consideration of the production plan for the units of output and the availability of additional technological support operations, which are strictly related to the number of the completed basic technological operations at the industrial unit. It is planned to develop a multiagent simulation model of the industrial unit's work and vehicle movements and optimise the values of the controlled model variables – the route of vehicle movement and the industrial unit' s work plan – using a modified genetic algorithm.

Acknowledgement

This work was performed under contract № 02.G25.31.0055 (project 2012-218-03-167).

References

[1] K. Aksyonov and A. Antonova, "Multiagent genetic optimisation to solve the project scheduling problem," Proc. ICCGI 2013: The Eighth International Multi-Conference on Computing in the Global Information Technology, Nice, France, pp. 237–242, July 2013.

[2] I. Okada, X. F. Zhang, H. Y. Yang, and S. Fujimura, "A random key-based genetic algorithm approach for resource-constrained project scheduling problem with multiple modes," Proc. Int. MultiConf. Engineers and Computer Scientists, vol. 1, Hong Kong, pp. 106–111, March 2010.

[3] M. Klimek, “A genetic algorithm for the project scheduling with the resource constraints,” Annals UMCS Informatica, vol. 10, issue 1, pp. 117–130, 2010.

[4] E. Sriprasert and N. Dawood, “Genetic algorithms for multi-constrained scheduling: an application for the construction industry,” Proc. CIB W78's 20th International Conf. Construction IT, Construction IT Bridging the Distance, CIB Report 284, Waiheke Island, New Zealand, pp. 341–353, April 2003.

[5] H. Abdel-Khalek, M. H. Sherif, A. M. el-Lacany, and Y. Abdel-Magd, “Financing – scheduling optimization for construction projects by using genetic algorithms,” World Academy of Science, Engineering and Technology, vol. 5, pp. 289–297, 2011.

[6] M. Karova, J. Petkova, and V. Smarkov, “A genetic algorithm for project planning problem,” Proc. Int. Scientific Conf. Computer Science, Krakov, Poland, pp. 647–651, June 2008.

[7] A. Brezuliani, L. Fira, and M. Fira, “A genetic algorithm approach for scheduling of resources in well-services companies,” International Journal of Advanced Research in Artificial Intelligence, vol. 1, no. 5, pp. 1–6, 2012.

[8] A. Dhingra and P. Chandna, “A bi-criteria M-machine SDST flow shop scheduling using modified heuristic genetic algorithm,” International Journal of Engineering, Science and Technology, vol. 2, no. 5, pp. 216–225, 2010.

[9] F.-C. Yang and W.-T. Wu, “A genetic algorithm based method for creating impartial work schedules for nurses,” International Journal of Electronic Business Management, vol. 10, no. 3, pp. 182–193, 2012.

[10] E. Osaba, R. Carballedo, and F. Diaz, “Simulation tool based on a memetic algorithm to solve a real instance of a dynamic TSP,” Proc. IASTED Int. Conf. Applied Simulation and Modelling, Napoli, Italy, pp. 27–33, June 2012.

[11] J. Xu and C. Feng, “Multimode resource-constrained multiple project scheduling problem under fuzzy random environment and its application to a large scale hydropower construction project,” The Scientific World Journal, vol. 2014, 2014. [Online]. Available from: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3914335/pdf/TSWJ2014-463692.pdf [Retrieved: May 2014]

[12] J. He and Z.-P. Wan, “Construction project scheduling problem with uncertain resource constraints,” Journal of Construction Engineering and Management ASCE, vol. 15, no.1, pp. 324–326, 2002.

[13] X. Zhang and X. Chen, “A new uncertain programming model for project scheduling problem,” An International Interdisciplinary Journal, vol. 15, no. 10, pp. 1–10, 2012.

[14] A. Csebfalvi, “A unified model for resource constrained project scheduling problem with uncertain activity durations,” International Journal of Optimization in Civil Engineering, vol. 2, no. 3, pp. 341–355, 2012.

[15] C. Artigues, R. Leus, and F. T. Nobibon, “Robust optimization for resource-constrained project scheduling with uncertain activity durations,” Flexible Services and Manufacturing Journal, vol. 25, no. 1–2, pp. 175–205, 2013.

[16] W.-N. Chen and J. Zhang, “Scheduling multi-mode projects under uncertainty to optimize cash flows: a Monte Carlo ant colony system approach,” Journal of Computer Science and Technology, vol. 27, no. 5, pp. 950–965, 2012.

[17] M. Brcic, D. Kalpik, and K.Fertalj, “Resource constrained project scheduling under uncertainty: a survey,” Proc. Central European Conference on Information and Intelligent Systems, Varazdin, Croatia, pp. 401–409, September 2012.

[18] D. Goldberg, Genetic Algorithms. Addison Wesley, 1989.

[19] I.G. Chernorutskii, Decision-making methods. St. Petersburg: BHV-Petersburg, 2005.

[20] M.-Y. Chen, H.-C. Tsai, and E. Sudjono, “Evaluating subcontractor performance using evolutionary fuzzy hybrid neural network,” International Journal of Project Management, no. 29, pp. 349–356, 2011.

[21] K. A. Aksyonov and A. S. Antonova, “Application of simulation and intelligent agents to solve project management problem,” International Journal of Computer Science Engineering and Information Technology Research, vol. 3, no. 1, pp. 321–330, 2013. [Online]. Available from: http://www.tjprc.org/view_archives.php?year=2013&id=14&jtype=2&page=3 [Retrieved: May 2014]

[22] B. Keller and G. Bayraksan, “Scheduling jobs sharing multiple resources under uncertainty: a stochastic programming approach,” IE Transactions, vol. 42, no. 1, pp. 16–30, 2009.

[23] J. Lehman and K. Stanley, “Exploiting open-endedness to solve problems through the search for novelty,” Proc. Eleventh Intern. Conf. Artificial Life (ALIFE XI), Cambridge, MA, pp. 329–336, August 2008.

[24] L. A. Zinchenko, V. M. Kureichik, and V. G. Redko (Eds.), Bionic Information Systems and their Practical Applications, Moscow, FIZMATLIT, 2011.

[25] W. L. Goffe, G. D. Ferrier, and J. Rogers, “Global optimization of statistical functions with simulated annealing,” Journal of Econometrics 60, pp. 65–99, 1994.

[26] K. Aksyonov et al., “Decision Support Systems Application to Business Processes at Enterprises in Russia,” in Efficient Decision Support Systems – Practice and Challenges in Multidisciplinary Domains, C. Jao, Ed. InTech, pp. 83–108, 2011. ISBN: 978-953-307-441-2. [Online]. Available from: http://www.intechopen.com/articles/show/title/decision-support-systems-application-to-business-processes-at-enterprises-in-russia. [Retrieved: May 2014]

[27] K. A. Aksyonov and N. V. Goncharova, Multi-agent resource conversion processes dynamic simulation, Yekaterinburg: USTU, 2006.

[28] K. A. Aksyonov, Research and development of tools for discrete resource conversion processes simulation, DPhil research paper, Ural State Technical University, Yekaterinburg, Russia, 2003.

[29] A. V. Andreichikov and O. N. Andreichikova, Intelligent information systems. Moscow: Finance and statistics, 2004.

[30] Modelling system Plant Simulation. The official web site. [Online]. Available from:

http://www.plm.automation.siemens.com/en_us/products/tecnomatix/plant_design/plant_simulation.shtml [Retrieved: May 2014]

[31] W. D. Kelton, J. S. Smith, D. T. Sturrock, and A. Verbraeck, Simio and Simulation: Modeling, Analysis, Applications. New York: McGraw-Hill, 2010.

[32] Real-time expert system G2. The official web site. [Online]. Available from: http://www.gensym.com/en/product/G2 [Retrieved: May 2014]

[33] Mutiagent modelling system AnyLogic. The official web site. [Online]. Available from: http://www.anylogic.com [Retrieved: May 2014]

[34] Mutiagent modelling system RepastJ. The official web site. [Online]. Available from: http://repast.sourceforge.net/index.html [Retrieved: May 2014]

[35] G. Rzevski, J. Himoff, and P. Skobelev, “MAGENTA technology: a family of multi-agent intelligent schedulers. International conference on multi-agent systems” Proceedings of Workshop on Software Agents in Information Systems and Industrial Applications 2 (SAISIA). Fraunhofer IITB, Germany, 2006. [Online]. Available from:

http://rzevski.net/06%20i-Scheduler%20Family.pdf
[Retrieved: May 2014]

[36] The official UML web site. [Online]. Available from: http://www.uml.org [Retrieved: May 2014]

[37] SQL Server language reference. [Online]. Available from: http://msdn.microsoft.com/en-us/library/ms166026(v=sql.90).aspx [Retrieved: May 2014]

[38] M. Wooldridge, "Intelligent agent: theory and practice," Knowledge Engineering Review, vol. 10 (2), 1995.

[39] J.P. Muller and M. Pischel, The Agent Architecture InteRRap: Concept and Application. Sarbrücken: German Research Centre for Artificial Intelligence (DFKI), 1993.

[40] FIPA ACL Message Structure Specification. [Online]. Available from: http://www.fipa.org/specs/fipa00061/SC00061G.html [Retrieved: May 2014]

[41] T. L. Saaty, Decision making with dependence and feedback: the analytic network process. University of Pittsburgh: RWS Publications, 1996.

[42] F. S. Hillier and G. J. Lieberman, Introduction to stochastic models in operations research. New York: McGraw-Hill, 1990.

[43] A. Chaleshtari and S. Shadrokh, "A branch and bound algorithm for resource constrained project scheduling problem subject to cumulative resources," Proc. International Conference on Information Management, Innovation Management and Industrial Engineering (ICIII), IEEE, vol. 1, Shenzhen, China, pp. 147–152, November 2011.

Review and Performance Analysis of Shortest Path Problem Solving Algorithms

Mariusz Głąbowski*, Bartosz Musznicki**, Przemysław Nowak*, and Piotr Zwierzykowski*

* Poznan University of Technology, Faculty of Electronics and Telecommunications,
Chair of Communication and Computer Networks, Poznań, Poland,
e-mail: mariusz.glabowski@put.poznan.pl, przemyslaw.nowak@inbox.com, piotr.zwierzykowski@put.poznan.pl
** INEA S.A., Poznań, Poland, e-mail: bartosz.musznicki@inea.com.pl

***Abstract*—The development of concepts derived from the generic approach to solving the problem of the shortest path resulted in numerous and various algorithms that appeared over the past decades. The studies on the most basic operation aimed at the determination of the shortest path between two given points in a graph (in other words, often a network) have resulted in sophisticated solutions designed for more and more demanding applications. Those include finding the sets of paths with the shortest distance between all pairs of nodes or searching for a shortest path tree. The aim of the present article is to give the reader an introduction to the problem of the shortest path and a detailed review of two groups of selected algorithms designed to solve particular problems. In the study described herein, different algorithms have been examined for their efficacy in their operation in directed graphs of different type represented in a well-defined data structure. The empirical simulation-based analysis proves that the performance varies among algorithms under investigation and allows to suggest, which methods ought to be used to solve specific variants of the shortest path problem and which algorithms should be avoided or used with caution.**

Keywords-shortest path; algorithms; review; performance; analysis

I. Introduction

This article is an extended version of a conference paper presented at AICT 2013, The Ninth Advanced International Conference on Telecommunications [1]. It introduces numerous additions, such as, more information on the problem of the shortest path, a detailed description of approaches and algorithms being tested, and a discussion of new simulation results. The foundations for the present review and performance analysis of selected algorithms are given by the research studies on shortest path problem solving using Ant Colony Optimization (ACO) metaheuristic approach [2]. It is just in the initial stage in the assessment of the potential in the applications of the ACO algorithms that the authors decided to start an in-depth analysis of those algorithms that represented a more traditional approach to the problem. As a result of the following investigations, relevant tests have been carried out. They are presented and compared in this article. It should be stressed that both well-known [3] and less commonly used algorithms are presented as long as they provide a possibility of finding the optimal solution having first satisfied some pre-defined initial requirements. Heuristic ACO algorithms have not been included in the presented evaluation for the simple reason that their operation does not, in fact, guarantee finding a solution that would always be optimal [4]. Moreover, the results obtained on the basis of ACO can be strongly dependent on the structure of the graph and there is no guarantee that any solution of any kind would be found at all [5].

The contents of the subsequent sections are arranged as follows. Section III shows two distinct approaches to the definition of the problem of the shortest path, lists some of its applications, and introduces the key assumptions, which shall be applied. Section IV is aimed at presenting the general types of shortest path problem solving algorithms. A detailed description and discussion of the two groups of algorithms that have been put to the analysis are presented in Section V. In addition, the relevance to and relationship with the shortest path tree is discussed. The data structure that represents the graphs under consideration is discussed in Section VI. Then, in Section VII, the graphs in which the simulations were carried out are described. The description is followed by Section VIII that will focus on the presentation and discussion of the results of the study. Finally, in Section IX, the article is summed up with conclusions.

II. Related work

In the process of careful investigation of publications related to the shortest path problem, numerous books and papers have been studied. The bulk of comparison papers are either directed at specific aspects and applications of the algorithms [6]–[9] or are focused on comparing new concepts with more classical methods [10], [11]. Some papers are concerned with asymptotic computational complexity [12]–[15], while other works are aimed at empirical computational complexity analysis of a number of algorithms based on implementation and simulation [7], [16]–[19]. In this paper, we decided to follow the latter approach to build this article upon experimental findings with respect to practical performance of a range of 12 closed-form complexity algorithms for solving shortest path problems that have not been compared before. The introduced homogeneous data structure

representing graphs under scrutiny is carefully discussed. Owing to the well-defined data structure, the results can be directly compared, which is critical in conclusive evaluation of the efficiency.

III. Problem of the shortest path

For the directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where N is the set of nodes (vertices) and A is the set of arcs (edges), we assign the cost a_{ij} to each of its edges $(i, j) \in \mathcal{A}$ (alternatively, this cost can be also called the length). We denote the biggest absolute value of an edge cost by C [see (1)]. For the resulting path $(n_1, n_2, \ldots, n_k)$, its length a_P can be expressed by (2).

$$C = \max_{(i,j)\in\mathcal{A}} a_{ij} \tag{1}$$

$$a_P = \sum_{i=1}^{k-1} a_{n_i n_{i+1}} \tag{2}$$

A path is called the shortest path if it has the shortest length from among all paths that begin and terminate in given vertices. The shortest path problem involves finding paths with shortest lengths between selected pairs of vertices. The initial (start) vertex will be designated as s, while the end (goal) vertex as t.

The problem of the shortest path can be also expressed differently [20]. If we go back to the original definition and define a_{ij} as the cost (and not the length), the problem can be reduced in its essence to a transmission of one flow unit between a pair of vertices as cheaply as it is possible. The problem is defined as follows: minimize (3a) limited by (3b) and (3c).

$$\sum_{(i,j)\in\mathcal{A}} a_{ij} x_{ij} \tag{3a}$$

$$0 \le x_{ij}, \qquad \forall\,(i,j) \in \mathcal{A} \tag{3b}$$

$$\sum_{\{j:(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j:(j,i)\in\mathcal{A}\}} x_{ji} = \begin{cases} 1, & \text{for } i = s \\ -1, & \text{for } i = t \\ 0, & \text{otherwise} \end{cases} \tag{3c}$$

This makes it possible to formulate the shortest path problem by defining a linear function that is analogous to the function that defines the minimum cost flow problem. To illustrate the comparison drawn above, let us assign a flow vector x that is described by

$$x_{ij} = \begin{cases} 1, & \text{for } (i,j) \in P \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

to a randomly selected path P from s to t.

Then x can be a solution to the problem (3), while the cost x is equal to the path length P. Hence, if vector x in the representation of formula (4) is the optimum solution to the problem (3), then the relevant corresponding path P is the shortest path.

A number of basic variants of the shortest path problem can be distinguished [21]:

- *finding the shortest path between a pair of vertices* For a given pair of vertices s and t the shortest path between them should be found. It should be mentioned here that so far no algorithm is known that solves this problem asymptotically in its worst case better than the best algorithm for the problem with one initial vertex.
- *finding the shortest paths with single initial vertex* For a given vertex s the shortest path between the vertex and each of the vertices $i \in \mathcal{N}$ is to be found.
- *finding the shortest paths with single end vertex* This is a reverse of the previous variant — the shortest path from each of the vertices $i \in \mathcal{N}$ to a given vertex t is to be found. By reversing each of the edges of the graph, the previous problem is obtained.
- *finding the shortest paths between all pairs of vertices* The shortest path between each pair of the vertices i and j that belong to $\mathcal{N}$ is to be found. A solution to this problem can be obtained by a solution of the shortest path problem with one initial vertex for each of the vertices in the graph.

In solving the problem of the shortest path we shall apply the following assumptions (which, in the case of some specific algorithms, may not be required).

- *The graph is a directed graph.* In the case of the undirected graph with non-negative weights, it is easy to transform it into a directed graph.
- *The graph does not include negative cycles.* The problem of the shortest path with negative cycles is $\mathcal{NP}$-hard (impossible to be presented using a polynomial algorithm).
- *There is a directed path between the pairs of vertices under consideration.*
- *Costs of the edge a_{ij} are integers* (this requirement applies to only some of the algorithms). In the case of the real costs of the edge, we can convert summations to integers multiplying them by an appropriately high number. Imaginary values would introduce unnecessary complications with their representations in computer-mediated activities.

The solution to the problem of the shortest path finds its application in a number of areas such as transportation or routing in communication networks [3], [22], [23] and is often related to searching for the shortest path tree in a graph.

It can be proved that the shortest paths from one node of a graph to all of the remaining nodes create a shortest paths tree [21], [25]. A characteristic feature of this tree is the fact that its root is formed from the initial (source) vertex, all of its edges are directed in the direction opposite to the vertex, and each path that can be created from the initial vertex to any other vertex is the shortest path to this vertex.

IV. Types of Shortest Path problem solving algorithms

The shortest path algorithms are characterized by certain common features — they are iterative, while their operation is based on assigning to particular vertices distance labels that are currently the best distances from the beginning of the path that is to be found. During the performance of these algorithms a set of valid vertices that can be taken into consideration is maintained. The method for a representation of this set may vary depending on a particular algorithm and can be representative for it. The difference in these algorithms is based on the method of updating the distance labels and a selection of a vertex expected to leave the mentioned set. Therefore, we can divide the shortest path algorithms into two groups [24]:

- *label-setting algorithms*
  This type of algorithms is characterized by a permanent setting of the distance label of one of the vertices in each iteration. This is equivalent to a single removal of a given vertex from the set of vertices under scrutiny. The most computationally complex part in these algorithms is mainly a selection, in each of the iterations, of a vertex with the lowest distance label from among the vertices that belong to the set of vertices under scrutiny. Algorithms from this group can be additionally applied only to acyclic graphs with defined (e.g., integer) edge lengths, or in the case when edges have non-negative lengths.
- *label-correcting algorithms*
  Unlike the algorithms of the previous group, algorithms of the type *label-correcting* treat all distance labels of vertices as temporary until the last iteration, whereupon all labels are set to the optimum value. This is translated then into a multiple addition of a vertex to the set of vertices under consideration and its multiple removal from the set. Due to the above, a choice of a vertex in each of the iterations is less computationally complex. Algorithms of this type can be used to solve all classes of the shortest path problem, including those with negative lengths of edges.

Label-setting algorithms can be viewed as a particular case of *label-correcting* algorithms. This means that *Label-correcting* algorithms can be used for solving more general cases of the problem. *Label-setting* algorithms for the case of non-negative lengths (costs) of edges have lower pessimistic complexity, which does not necessarily have to translate into better expected (average) complexity. All discussed *label-correcting* algorithms achieve identical pessimistic computational complexity. Differences in effectiveness of the algorithms can be seen in their practical applications or with particular graphs.

By taking into consideration practical applications of the algorithms under study, a numbers of factors are in favour of *label-correcting* algorithms. These algorithms are more elastic and, in consequence, can be better adjusted to making use of additional initial data for a given graph. They can be also better adjusted to a problem in providing a solution to which they have been used — it is even possible in some cases to make distance labels set only once (for one vertex to be added and removed from the set of considered vertices). This equalizes the most important advantage of *label-setting* algorithms.

In practice, graphs that have edges with negative costs are rare, while for these cases good *label-correcting* algorithms have better expected complexity than *label-setting* algorithms. It is so because, beside the required $O(A)$ number of operations for the algorithms of both types that is needed to check each of the edges at least once, the *label-setting* algorithms require approximately additional operations with their number proportional to N, while the number of operations of additional *label-correcting* algorithms approximately increases linearly with A. For rare graphs the ratio of additional operations of both groups of algorithms is much favourable for *label-correcting* algorithms, whereas for dense graphs, for *label-setting* algorithms.

V. Algorithms for solving Shortest Path problems

The following subsections of this section focus on the algorithms for a determination of the shortest paths between a given single initial vertex and all the remaining vertices of the graph.

The algorithms solving shortest path problems that are briefly discussed in the following subsections have been evaluated through efficiency analysis. Each of the algorithms has particular features that eventually lead to their differences in their properties and performance. On account of their possible applications, the algorithms have been, in turn, divided into two categories.

A. Single-Source Shortest Paths problem

The following subsections of this section focus on algorithms for a determination of the shortest paths between a given single initial vertex and all the remaining vertices of the graph.

1) Generic algorithm: The operation of the generic algorithm [20] is based on iterative checking of edges from the vertex under consideration i and on label setting for vertex j, in which a given edge terminates, to $d_j = d_i + a_{ij}$, in the case when $d_j > d_i + a_{ij}$. To store the vertices that are to be checked, the list V is used, called *candidates list*. The way vertices are stored in this list, as well as the method determining the addition and the retrieval of vertices to and from it, is frequently the major factor that distinguishes individual algorithms under consideration. In the case of the generic algorithm, the candidates list is a *FIFO* queue in

which operations of additions and retrieval of a vertex to the end of it or from its head, respectively, are performed.

The algorithm starts to check from the initial vertex s, with initial conditions defined by (5).

$$\begin{aligned} V &= \{s\}, \qquad d_s = 0 \\ d_i &= \infty, \qquad \forall\, i \neq s \end{aligned} \tag{5}$$

The algorithm checks individual edges of the initial vertex and, if an appropriate condition is satisfied, sets the labels of the vertex in which a given edge terminates, adding it to the candidates list if it is not already there. The procedure is then repeated until the list of candidates is empty.

During the performance of the algorithm labels are monotonically non-increasing and if $d_i < \infty$, then vertex i has appeared on the candidates list V at least once.

2) Dijkstra's algorithm: Dijkstra's algorithm is presumably the best known algorithm for finding the shortest path in the directed graph [26]. The method is an algorithm of the type *label-setting*, which means that once considered vertex does not appear again on the list of candidates, while its label, once it is set, is ultimate and denotes the shortest distance from the initial vertex to this vertex.

The initial conditions are illustrated in (5), while an additional constraint is non-negativity of the length of the edge (6).

$$a_{ij} \geq 0 \tag{6}$$

The basic difference between this algorithm and the generic algorithm is the way in which vertices are drawn from the candidates list — the selected vertex is the vertex that has the smallest label from all available vertices in the list:

$$d_i = \min_{j \in V} d_j \tag{7}$$

This causes the vertex with its label set, as well as all vertices that are in the path from the initial vertex to this particular vertex, to have the minimum value of the label and to not be added again to the candidates list. The number of iterations of the algorithm is equal to the number of vertices N. During each iteration, two operations are performed — the choice of a vertex from among all vertices on the list of candidates, and scanning and, should the need arises, setting of distance labels. The choice of a vertex in its worst case requires $O(N)$ operations, which in conjunction with the number of iterations gives $O(N^2)$ operations. Checking of the labels is performed A times, since in each iteration the algorithms checks all edges that start in the vertex under scrutiny, whereas each vertex is considered only once. $O(A)$ is not taken into account because it is far smaller than $O(N^2)$. Therefore, the total number of operations that the Dijkstra's algorithm needs to perform to solve the shortest path problem is $O(N^2)$.

3) Dijkstra's algorithm using a heap: It is not possible to decrease the number of operations that are performed in order to check labels, because this would not make it possible to guarantee the optimum solution finding — each edge has to be checked at least once. A selection of an optimum data structure that represents the candidates list makes it possible, in turn, to reduce significantly the computational complexity of the operation of the selection of a vertex from the candidates list [27]. Here, heaps (also known as priority queues) can serve ideally the purpose. Using Fibonacci heap we can solve the shortest path problem using Dijkstra's algorithm and performing $O(A + N \log N)$ operations.

4) Dial's algorithm: Another way to reduce the number of operations accompanying the selection of a vertex from the candidates list is a division of the list into buckets [28]. Each bucket B_k stores only vertices with a given label k. This causes lengths of edges to have to be integers and non-negative. When this is the case, labels can take on values from 0 to $(N-1)C$. This gives $(N-1)C+1$ of different values of the labels and, at the same time, buckets that have to be scanned in increasing order until the first non-empty bucket is found. After a given vertex is checked, it is removed from the bucket and scanning in the next iteration starts with this particular bucket. As a result, once emptied bucket is not checked again any more. It happens so because the currently checked vertex always has the lowest (smallest) label from among any other vertices in the buckets and, since lengths of edges are non-negative, while setting labels in a given iteration none of the label will be set to a value that is lower than the value of the label of the vertex that is being checked.

A good structure for the implementation of buckets is the two-way list. The list allows all operations (checking whether a bucket is empty, addition of a vertex and its removal from the bucket) to be performed in time $O(1)$. Taking it all into consideration, the choice of a vertex requires $O(NC)$ operations, which, after taking into account $O(A)$ operations for checking and setting labels, results in the computational complexity of Dial's algorithm being $O(A + NC)$. What is crucial to understand, is that the bucket deletion and insertion operations require linear time and not more than NC buckets need to be examined by the procedure [17]. The higher the absolute value of an arc cost C, the more operations need to be performed by the algorithm, and thus, the performance gain related to the usage of buckets dramatically diminishes. Therefore, for small values $C \ll N$, Dial's algorithm performs very well in practice.

5) Bellman-Ford algorithm: The Bellman-Ford algorithm belongs to algorithms of the *label-correcting* type, i.e., the ones treating all labels for vertex distances as temporary until the last iteration, after which all labels are set to optimum values [29]. This algorithm provides a possibility

to solve the shortest path problem in graphs with negative lengths of edges. In the case when a negative cycle is found, the algorithm yields false return as the result of its operation. Because of this particular method of operation of the algorithm, the candidates list is not required. The initial conditions are in accordance with (5), though with the omission of the list V. The algorithms checks all edges of the graph $N-1$ times, which allows the minimum labels in the graph to be propagated. In its final stage, the algorithm checks whether any of the labels is non-optimum — this situation happens only in the case of the occurrence of a negative cycle in the graph, which is reported by the algorithm by yielding false returns. This algorithm makes $N-1$ iterations in which it checks A edges. Its computational complexity is then equal to $O(NA)$.

6) D'Esopo-Pape algorithm: The D'Esopo-Pape algorithm uses the candidates list in the form of a queue [30]. Vertices that are to be checked are always retrieved from the head of the list. However, the place a given vertex is added to in the candidate list depends on whether the vertex has already been placed in this list. If this is the case, it is added to its head, otherwise — to the end of the list. This is caused by the fact that a modification of the label of vertex i can be followed by a modification of vertices j such that $(i,j) \in \mathcal{A}$. A quicker updating of the vertices in which the edge that starts in the considered vertex terminates effects in the optimum of the solution to be quicker achieved. Such an operation of the algorithm results in its good results in practice. There are instances, however, when the algorithm completely cannot cope with, and the number of additions of some vertices to the candidates list is non-polynomial.

7) SLF algorithm: The Small Label First algorithm (*SLF*) seeks to manage the candidates list in such a way as to make vertices with small labels located as close to the head of the list as possible [31]. The reason for this operation is the fact that the smaller the label of a vertex that is retrieved from the candidates list, the lower the probability that this vertex will be forwarded to the list once again. This algorithm, just as the two following algorithms, attempts to reach the characteristic operation of Dijkstra's algorithm with a lower computational outlay. The algorithms are designed for graphs with non-negative edges, though they also operate otherwise (there is no guarantee then that they will perform better).

In each of its iteration the algorithm *SLF* checks the vertex that is placed at the head of the candidates list. The place of the addition of a vertex after its label has been changed depends on the value that is taken on by the label. If the vertex label of the vertex that is to be added to the candidates list is lower or equal to the label of the vertex that is currently at the head of the list, this vertex is added as the first in the list. Otherwise, the vertex is added at the very end of the candidates list.

8) LLL algorithm: The Large Label Last algorithm (*LLL*) attempts to achieve the operation that is similar to that of the previous algorithm using a specific method for the retrieval of vertices from the candidates list [32]. The addition of vertices to the candidates list is not defined in any way. However, the method for their retrieval from the list is defined. Each time when a vertex is to be taken from the list, the average value of the labels of the vertices in the list is calculated. Then, the label of the vertex that is at the head of the list is compared with this average. If the label of the vertex is higher than the average, the vertex is moved to the end of the list. Otherwise, the vertex is returned as the one that has to be considered in this iteration.

9) SLF/LLL algorithm: The *SLF*/*LLL* combines the *SLF* algorithm method for the addition of vertices to the candidates list and the *LLL* algorithm method for their retrieval from the list [20]. The *SLF*/*LLL* algorithm requires a lower number of iterations to solve the shortest path problem than the algorithms it combines. This is done, however, at the cost of the increased number of necessary calculations. To speed up the process, parallel computing methods can be applied.

B. All-Pairs Shortest Path problem

The following subsections present algorithms that are dedicated to finding the shortest paths between all pairs of vertices.

1) The doubling algorithm: The algorithm's operation is based on iterative calculation of the shortest paths for all vertices composed of an increasing number of edges [33]. It starts with paths that are composed of just one edge, and then checks whether paths that are composed of two edges would not be shorter. This operation is then repeated until all paths that are composed of $N-1$ edges are checked. This procedure has some similarity with matrix multiplication [25].

Matrices that are used in the algorithms $D^m = \{d_{ij}^m\}$ and $Pred^m = \{pred_{ij}^m\}$ have the initial values (8a) and (8b), respectively.

$$d_{ij}^1 = \begin{cases} 0, & \text{for } i = j \\ a_{ij}, & \text{for } (i,j) \in \mathcal{A} \\ +\infty, & \text{otherwise} \end{cases} \tag{8a}$$

$$pred_{ij}^1 = \begin{cases} 0, & \text{for } i = j \\ i, & \text{for } (i,j) \in \mathcal{A} \\ 0, & \text{otherwise} \end{cases} \tag{8b}$$

In its simplest case, the matrix D^1 would be multiplied by itself $N-2$ times to take into account paths that have $1, 2, \ldots, N-1$ edges. The matrices that correspond to particular iterations would be as in (9), though such a case

would effect in the complexity at the level $\Theta(N^4)$.

$$\begin{aligned} &D^1 \\ &D^2 && = D^1 \cdot D^1 \\ &D^3 && = D^2 \cdot D^1 \\ &&& \vdots \\ &D^{(N-1)} && = D^{(N-2)} \cdot D^1 \end{aligned} \tag{9}$$

The knowledge of the values of all matrices D is not, however, necessary — it is the matrix $D^{(N-1)}$ that needs special attention. Hence, the doubling algorithm, instead calculating successive matrices D, calculates only its powers of 2 [see (10)].

$$\begin{aligned} &d_{ij}^{2^m} = \min_k\{d_{ik}^{2^{(m-1)}} + d_{kj}^{2^{(m-1)}}\}, \\ &i,j,k \in \mathcal{N},\ m = 1,2,\ldots,\lceil \log_2(N-1) \rceil \end{aligned} \tag{10}$$

Bearing in mind the fact that a path that is composed of more than $N-1$ edges cannot be shorter than the shortest path, we know that $D^n = D^{(N-1)}$ for all $n \geq N-1$. In such a case we have a sequence of matrices [see (11)].

$$\begin{aligned} &D^1 \\ &D^2 && = D^1 \cdot D^1 \\ &D^4 && = D^2 \cdot D^2 \\ &&& \vdots \\ &D^{2^{\lceil \log_2(N-1) \rceil}} && = D^{2^{\lceil \log_2(N-1) \rceil - 1}} \cdot D^{2^{\lceil \log_2(N-1) \rceil - 1}} \end{aligned} \tag{11}$$

This gives the ultimate computational complexity of the algorithm equal to $\Theta(n^3 \log_2 N)$.

2) Floyd-Warshall algorithm: The Floyd-Warshall algorithm obtains what the previous algorithm was capable of, using a different approach and achieving at the same time lower computational complexity equal to $\Theta(N^3)$ [34], [35]. The algorithm analyses the internal vertices of the path $P = (i, n_1, n_2, \ldots, n_k, j)$, i.e., those that are neither the initial (original) vertex nor the goal vertex. For the given path P, these are the vertices from the set $\{n_1, n_2, \ldots, n_k\}$.

Assuming that $\mathcal{N} = \{1, 2, \ldots, N\}$, for a certain k, let us consider the sub-set $\{1, 2, \ldots, k\}$ and all paths from i to j whose internal vertices belong to this sub-set, for each pair of the vertices $i, j \in \mathcal{A}$. From among all the paths we denote the shortest path as P. The assumption is that the graph does not include non-negative cycles and, thus, this path is a simple path (does not have repeated vertices or edges). We analyse this path against the shortest paths from i to j that have the set of internal vertices limited to the sub-set $\{1, 2, \ldots, k-1\}$.

Depending on whether k is an internal vertex of path P, we can draw the following conclusions. If k is not an internal vertex of the path P, then it means that its internal vertices are limited to the sub-set $\{1, 2, \ldots, k-1\}$. Then, P is the shortest path also when the set of its internal vertices is equal to $\{1, 2, \ldots, k\}$. Intuitively, this means that an expansion of the set of internal vertices does not change the shortest path. If, however, k is the internal vertex of the path P, we divide it into two paths P_1 from i to k and P_2 from k to j. Both paths are the shortest paths, while the set of their vertices is limited to the sub-set $\{1, 2, \ldots, k-1\}$, since vertex k is the goal vertex of path P_1 and the initial vertex of path P_2. This means, in turn, that by dividing the path into two shortest paths we can limit the set of their internal vertices. In both cases we obtain the shortest path for a given k using the shortest path for $k-1$.

The matrices used in this algorithm are exactly the same as the matrices used in the doubling algorithm. The initial values for these matrices are identical with equation (5) with the only difference that, instead d_{ij}^1 and $pred_{ij}^1$, we define respectively d_{ij}^0 and $pred_{ij}^0$.

On the basis of the earlier considerations and initial conditions we obtain the recurrent formula (12).

$$d_{ij}^k = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}, \qquad i,j,k \in \mathcal{N} \tag{12}$$

The formula illustrates precisely in how the length of the shortest path is dependent on whether k is its internal vertex and whether uses the values obtained for $k-1$.

3) Johnson's algorithm: For sparse graphs (i.e., those in which the number of edges is far lower than N^2) it is possible to improve the process of calculation of the shortest paths between all pairs of vertices using Johnson's algorithm [36]. For this purpose, the two algorithms discussed earlier, i.e., the Bellman-Ford algorithm and Dijkstra's algorithm (most favourably in its form with a heap), are used. Therefore, the Johnson's algorithm has a number of particular properties and limitations of both algorithms. It can determine whether a graph includes a negative cycle (just like the Bellman-Ford algorithm) and requires non-negative lengths of the edge (like Dijkstra's algorithm). Getting round this limitation is possible thanks to an appropriate transformation of the graph presented in (13).

In the initial stage of the Johnson's algorithm, the graph is being modified in order to get rid of edges with negative lengths. Such a transformation has to guarantee additionally that the shortest paths in the graph do not change. To achieve that, the graph is being added the additional vertex s that is to be the initial vertex for the Bellman-Ford algorithm. To each of the earlier vertices the edge that starts in s with the length equal to 0 is also added.

$$\begin{aligned} \mathcal{G}' &= (\mathcal{N}', \mathcal{A}') \\ \mathcal{N}' &= \mathcal{N} \cup \{s\}, \qquad s \notin \mathcal{N} \\ \mathcal{A}' &= \mathcal{A} \cup \{(s,j) : j \in \mathcal{N}\} \\ a' &= a, \qquad a'_{sj} = 0, \qquad \forall\, j \in \mathcal{N} \end{aligned} \tag{13}$$

Thus created graph $\mathcal{G}'$ has no paths that would include the vertex s except those that start in it, and includes negative cycles only when the graph $\mathcal{G}$ has included such cycles. If the graph $\mathcal{G}$ does not include negative cycles, then, after the execution the Bellman-Ford algorithm on the graph $\mathcal{G}'$ with the vertex s as the initial vertex, we obtain the vector h that

defines the lengths of the shortest paths in this graph. The vector is then used to modify the lengths of edges in such a way as not to make them non-negative, in line with (14).

$$a'_{ij} = a'_{ij} + h_i - h_j, \qquad i,j \in \mathcal{A}' \tag{14}$$

Then, for each vertex i that belongs to $\mathcal{A}$ Dijkstra's algorithm is applied to calculate all the shortest paths that start in it. After the calculation of the lengths of the paths that start in a given vertex, they are modified in such a way as to reflect and correspond to the lengths of paths in the original graph [see (15)].

$$d_{ij} = d'_j + h_j - h_i, \qquad i,j \in \mathcal{A} \tag{15}$$

In this way, the matrix $D = \{d_{ij}\}$ is obtained. The matrix includes the lengths of the paths between all pairs of the vertices.

In the Johnson's algorithm, Dijkstra's algorithm is performed N times and it is the latter algorithm that significantly influences the computational complexity of the whole algorithm. If we choose to apply the implementation of Dijkstra's algorithm with Fibonacci heap, then we are obliged to perform $O(NA + N^2 \log N)$ operations to calculate the shortest paths between all the pairs of vertices in a sparse graph. Using a binary heap would result in an increase in the number of necessary operations to $O(NA \log N)$.

VI. Data structure representing graphs

To represent graphs during the simulation, a double associative adjacency array was used. This structure is composed of two associative arrays — one (external), representing vertices from which edges originate, and the other (internal) representing all vertices, which edges for a given row of the first matrix (table) join. Such a representation provides an opportunity to minimize shortcomings of typical structures, such as the list of edges or the adjacency matrix, providing at the same time appropriately low computational complexity for individual operations. The applied structure makes it possible to store additional information about edges, e.g., weights or costs. A homogeneous method for the projection (mapping) of graphs for all simulated algorithms ensures further comparability of the results of simulations.

The operation of the structure may differ depending on the implementation of the associative array and is dependable on the programming language used if embedded structures are used. The most crucial operation is the operation of checking whether a given key is in the array, hence structures that handle this best, e.g., hash tables or self-balancing binary search trees, are applied. Additionally, we can adjust the operation of the double associative adjacency array for our particular needs and thus make it possible, for example, to sort vertices in the internal array, which a given edge joins using a heap.

For a graph with edge weights, the double, associative adjacency array T_{2asoc} can be written as follows:

$T_{2asoc} = T_{ext}$	external array
$T_{2asoc}[i] = T_{ext}[i] = T_{int_i}$	internal array for edge coming out from vertex i
$T_{2asoc}[i][j] = T_{int_i}[j] = a_{i,j}$	edge weight (i,j)

Therefore, the graph in Fig. 1 will be mapped in the following way:

$$\begin{array}{lll}
T_{2asoc}[1] & = T_{int_1} & \\
T_{2asoc}[1][3] & = T_{int_1}[3] & = a_{1,3} \\
T_{2asoc}[1][5] & = T_{int_1}[5] & = a_{1,5} \\
T_{2asoc}[2] & = T_{int_2} & \\
T_{2asoc}[2][1] & = T_{int_2}[1] & = a_{2,1} \\
T_{2asoc}[2][2] & = T_{int_2}[2] & = a_{2,2} \\
T_{2asoc}[3] & = T_{int_3} & \\
T_{2asoc}[3][2] & = T_{int_3}[2] & = a_{3,2} \\
T_{2asoc}[3][5] & = T_{int_3}[5] & = a_{3,5} \\
T_{2asoc}[4] & = T_{int_4} & \\
T_{2asoc}[4][1] & = T_{int_4}[1] & = a_{4,1} \\
T_{2asoc}[4][3] & = T_{int_4}[3] & = a_{4,3} \\
T_{2asoc}[5] & = T_{int_5} & \\
T_{2asoc}[5][4] & = T_{int_5}[4] & = a_{5,4}
\end{array}$$

Characteristic features of the structure:

- required memory: $O(N + A)$
- effective memory complexity for directed sparse graphs
- effective execution of graph algorithms that require to reach all vertices adjacent to a given vertex (logarithmic complexity)
- capacity of remembering parallel edges (all edges between the same pair of verices)
- effective execution of checking whether the graph includes a given edge (logarithmic complexity)
- effective execution of addition and removal of edges of a graph (logarithmic complexity)
- possibility of a substitution of the internal associative table with some other structure, e.g., in order to sort

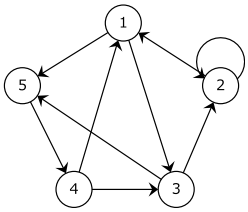


Figure 1. Exemplary directed graph

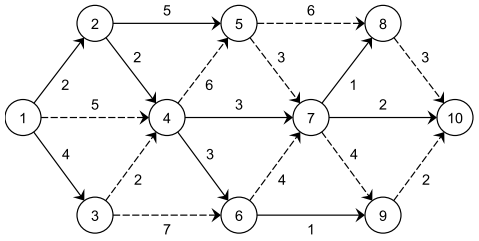


Figure 2. Manually created *custom 1* graph in which the edges marked with the solid line create a shortest paths tree with the root in node 1

TABLE I. STRUCTURE OF THE GRAPHS USED IN THE SIMULATION

graph	vertices	edges	
		number	lengths
custom 1 (c1)	10	19	$\langle 1, 7\rangle$
custom 2 (c2)	20	43	$\langle 0, 9\rangle$
multistage 1 (ms1)	52	420	$\langle 1, 9\rangle$
multistage 2 (ms2)	86	249	$\langle 1, 10\rangle$
random 1 (r1)	25	125	$\langle 1, 9\rangle$
random 2 (r2)	100	628	$\langle 1, 20\rangle$

vertices in which a given edge terminates by the weight of the edge (e.g., using binary, Fibonnaci, binomial or Relaxed heap)

- fairly complicated in its execution

VII. GRAPHS USED IN THE SIMULATION

To examine the efficiency and performance of the algorithms during their operation in different graphs, directed graphs constructed manually and those that were generated pseudo-randomly were used. To discuss the results, the 6 representative graphs described in Table I were selected. Graph *custom 1* shown in Fig. 2 was created manually from 10 vertices that were joined together by 19 edges. The *custom 2* graph was created manually as well and consists of 20 vertices and 43 edges. Another two graphs that were used in the tests are the graphs that are characteristic for a multi-stage shortest path problem. An exemplary graph is presented in Fig. 3. The first multi-stage graph used in the tests, *multistage 1*, has 5 stages, each having 10 vertices. The lengths of edges were generated randomly from within the interval $\langle 1, 9\rangle$. The second multi-stege graph, *multistage 2*, has 30 stages, each having 3 vertices, and therefore, the structure comprises 249 edges. The *random 1* graph was generated randomly, without loops, and with 5 edges coming out of each of the vertices. The last graph under scrutiny, *random 2*, was also generated randomly, without loops, but with 3 to 10 edges coming out of each of the vertices.

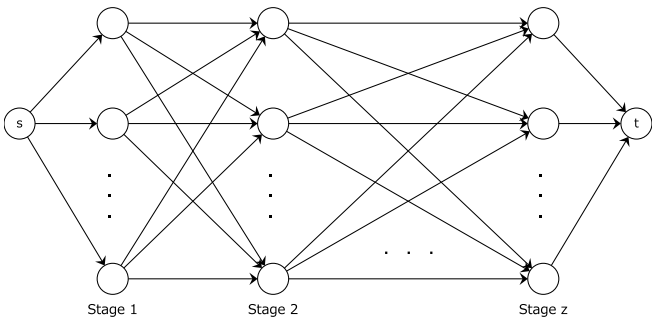


Figure 3. A general structure of a multi-stage graph

VIII. RESULTS OF THE SIMULATIONS OF THE ALGORITHMS

All the tests were carried out in a simulation environment prepared in C# programming language. In order to achieve reliable results, each algorithm was performed 100 times for each of the graphs. To eliminate the influence of the simulation environment, extreme results were rejected and then the average of the remaining results was calculated.

Table II shows the running (execution) times of the algorithms tested for the graphs discussed in Section VII. The results are divided into two groups — algorithms solving Single-Source Shortest Paths problem (*SSSP*) and algorithms solving All-Pairs Shortest Path problem (*APSP*). The best results for each graph are highlighted in bold text, and the worst are in italics.

The graph *custom 1* was solved by all *SSSP* algorithms in almost identical times. Of all the algorithms only two deserve a mention here — Dijkstra's algorithm with a heap (that operated within the longest time), and *SLF* (that solved the problem slightly quicker than the rest). The results that were very similar to that of the *SLF* algorithm were also shared by Dijkstra's algorithm, Dial's algorithm and the *LLL* algorithm. From the group of the *APSP* algorithms, it was the Floyd-Warshall algorithm that fared the best, being less than twice as long as the *SSSP* algorithms. The remaining algorithms needed about twice as much time to find all paths.

The next graph the simulations were performed on, *custom 2*, was salved in the group of *SSSP* algorithms in the shortest time by the *SLF* algorithm. Bellman-Ford algorithm and Dijkstra's algorithm with a heap were the slowest ones. The remaining algorithms finished in quite similar times. In the case of *APSP* algorithms Johnson's was the fastest, being slightly better than Floyd-Warshall algorithm and over 3 times faster than the doubling algorithm.

The first graph characteristic for the multi-stage shortest path problem *multistage 1* brought a significant increase in

TABLE II. COMPARISON OF RUNNING TIMES FOR THE ALGORITHMS SOLVING THE SHORTEST PATH PROBLEM IN MICROSECONDS

algorithm	graph					
	c1	c2	ms1	ms2	r1	r2
generic	112	127	312	247	163	603
Dijkstra	100	122	324	258	148	405
DijkstraHeap	*146*	176	466	323	200	518
Dial	104	128	322	282	172	395
Bellman-Ford	119	*217*	*3252*	3097	*511*	*8526*
D'Esopo-Pape	113	147	1260	*4171*	239	1222
SLF	**96**	**111**	**262**	**236**	**143**	**376**
LLL	102	121	336	274	155	422
SLF/LLL	112	132	318	288	161	431
doubling alg.	324	*2594*	*47678*	*233517*	*4756*	*364714*
Floyd-Warshall	**184**	1031	16045	70420	**2057**	112880
Johnson	*418*	**879**	**9309**	**12970**	2959	**41904**

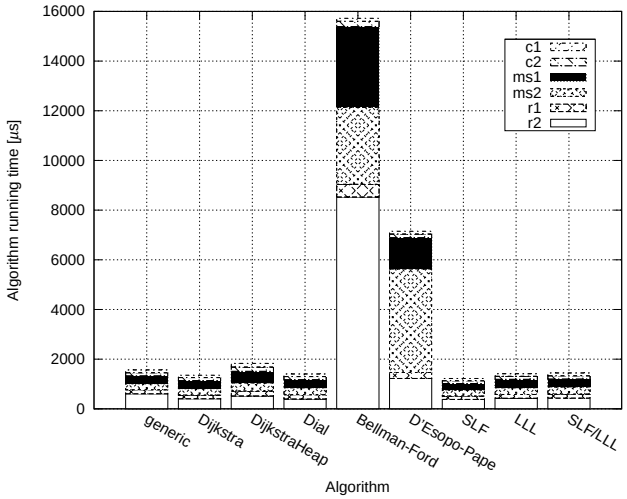


Figure 4. Chart of running (execution) times of the algorithms solving the shortest path problem with one initial vertex (*SSSP*)

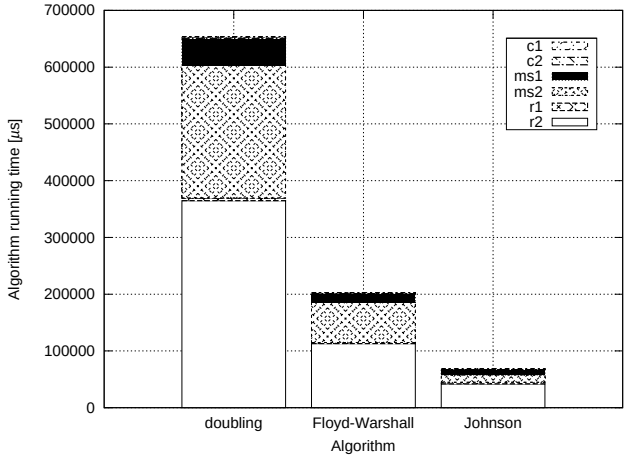


Figure 5. Chart of running (execution) times of the algorithms solving the shortest path problem between all pairs of vertices (*APSP*)

differences between *SSSP* algorithms. Again, the *SLF* algorithm was the quickest, whereas Bellman-Ford and D'Esopo-Pape algorithms handled the problem the worst. Except Dijkstra's algorithm with a heap, which was performing slightly longer than the rest, the remaining algorithms had similar running times. This situation for the *APSP* algorithms was exactly as in the case of the previous graph — Johnson's algorithm was the quickest and the doubling algorithm was the slowest, while the distance between Johnson's and Floyd-Warshall algorithms increased.

The *SLF* proved to be the quickest for the *multistage 2* graph, and hence, it was faster than the generic algorithm and the *SLF/LLL* algorithm that came second and third, accordingly. The D'Esopo-Pape and the Bellman-Ford algorithms performed the worst and were 10 to 14 times slower than other algorithms. In the group of *APSP* algorithms the Johnson's algorithm took the shortest time to solve the problem, about 5 times faster than Floyd-Warshall algorithm and about 18 times faster than the doubling algorithm.

Another graph under consideration, *random 1*, was solved the quickest in the *SSSP* mode by the *SLF* algorithm, with Dijkstra's algorithm as the runner up and the Bellman-Ford and the D'Esopo-Pape algorithms well behind the two. The latter two were the worst as compared to all involved *SSSP* algorithms. This time, the quickest *APSP* algorithm was the Floyd-Warshall algorithm. Johnson's algorithm performed slightly worse, while the doubling algorithm was the worst (the longest) of the lot.

The last graph, *random 2*, consists of the highest number of edges. However, it poses no problem for the *SLF* algorithm to solve it in the shortest time in the group of *SSSP* algorithms. Dial and Dijkstra's algorithms gave good results as well. Bellman-Ford algorithm operated clearly longer as compared to the rest of the algorithms. The D'Esopo-Pape algorithm was also slow once again. If we take a look at the *APSP* algorithm the results are analogical to the most of the prior results. Johnson's algorithm is the most time-efficient with the Floyd-Warshall algorithm being almost 3 times slower and the doubling algorithm to be far behind, and thus, the last one in the race.

The procedures that solve the *SSSP* problem best include the *SLF* algorithm, that had the shortest times for each tested graph, and Dijkstra's algorithm, that always performed with a quite similar time. The *LLL* and the *SLF/LLL* algorithms performed very well and did not generate solutions over times that differ much from those provided by the quickest algorithm. The generic algorithm and Dial's algorithm performed slightly better or slightly worse depending on the chosen graph. Dijkstra's algorithm with a heap had some problems and, instead of performing quicker than Dijkstra's algorithm, was slower. In this particular case, this can be most probably explained by the missing optimization of the heap that formed the base for the algorithm. Undoubtedly, however, an improvement in the running time during, which solutions are provided is still possible. At least, an improvement in the execution time needed for the algorithm to generate solutions is possible. As it is clear from Fig. 4, for the Bellman-Ford and D'Esopo-Pape algorithms, the worst case occurs far too often, which may result from both non-optimal implementation and from the possibility of their operation on graphs that were unsuitable for them. The D'Esopo-Pape algorithm was much quicker to solve graphs, but irrespective of the fact it underperformed far too much as compared to the rest of the algorithms. Underperformance of the latter group of algorithms is particularly visible in graphs that have a higher number of edges, which results from the assumptions, as they were, that served as a basis for their design.

The *APSP* algorithms were decidedly varied across dif-

ferent performance dimensions, in particular in relation to the time necessary to generate results, which is clearly shown in Fig. 5. The doubling algorithm was the slowest and performed several times slower than the competitors. The Floyd-Warshall algorithm was the fastest for 2 graphs, while for the remaining graphs it was in second place. The differences in the time needed for graphs to be solved are in its case significant as compared to Johnson's algorithm that overall turned out to be the fastest one.

IX. Conclusion

This article provides a detailed presentation of 12 algorithms solving the shortest path problem and presents an analysis of their performance. The study showed that in a prepared simulation environment that ensured directed graphs of different type to be provided, the weakest aggregated time results from among all the available algorithms solving the Single-Source Shortest Paths problem were those of, in the descending order, the Bellman-Ford and the D'Esopo-Pape algorithms. The fastest algorithm was Small Label First algorithm, slightly faring better than Dijkstra's algorithm. From the pool of the algorithms dedicated for All-Pairs Shortest Path problem, the doubling algorithm performed decidedly worst, while the best results were those of Johnson's algorithm.

In addition to the presentation of run-time relationships between the algorithms, the study indicates the importance and significance of an appropriate choice of a method destined to solve the problem that would be the most efficient for a type of the graph structure that is to be used. Moreover, it is worthwhile to remember that details concerning the implementation as well as the architecture of the structures for the representation of data can significantly influence the performance of an algorithm.

Future work could focus on conducting experiments in larger graphs, including those obtained by using Internet-like topology generators, and in the structures that reflect the relationships in the society or complex databases. The operation in graphs with a power-law distribution of node degrees may prove to be interesting and useful as well. Furthermore, the performance measured and evaluated in FLOPS (FLoating point Operations Per Second), instead of average running time, may give a new and broader insight into the problem.

Acknowledgement

This work has been partially supported by a grant from Switzerland through the Swiss Contribution to the enlarged European Union (PSPB-146/2010, CARMNET).

References

[1] M. Głąbowski, B. Musznicki, P. Nowak, and P. Zwierzykowski, "Efficiency evaluation of shortest path algorithms," in Proceedings of AICT 2013, The Ninth Advanced International Conference on Telecommunications, Rome, Italy, 23–28 June 2013, pp. 154–160.

[2] M. Głąbowski, B. Musznicki, P. Nowak, and P. Zwierzykowski, "Shortest path problem solving based on ant colony optimization metaheuristic," International Journal of Image Processing & Communications, Special Issue: Algorithms and Protocols in Packet Networks, vol. 17, no. 1–2, 2012, pp. 7–17.

[3] B. Y. Wu and K.-M. Chao, Spanning Trees and Optimization Problems. USA: Chapman & Hall/CRC Press, 2004.

[4] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: overview and conceptual comparison," ACM Computing Surveys, vol. 35, no. 3, September 2003, pp. 268–308.

[5] M. Głąbowski, B. Musznicki, P. Nowak, and P. Zwierzykowski, "An in-depth discussion of challenges related to solving shortest path problems using ShortestPathACO based algorithms," in Information Systems Architecture and Technology; Knowledge Based Approach to the Design, Control and Decision Support, J. Świątek, L. Borzemski, A. Grzech, and Z. Wilimowska, Eds. Wrocław, Poland: Oficyna Wydawnicza Politechniki Wrocławskiej, 2013, pp. 77–88.

[6] R. Vasappanavara, E. V. Prasad, and M. N. Seetharamanath, "Comparative studies of shortest path algorithms and computation of optimum diameter in multi connected distributed loop networks," Multi-, Inter-, and Trans-disciplinary Issues in Computer Science and Engineering, vol. 2, no. 1, January 2006, pp. 62–67.

[7] B. V. Cherkassky, L. Georgiadis, A. V. Goldberg, R. E. Tarjan, and R. F. Werneck, "Shortest path feasibility algorithms: an experimental evaluation," ACM Journal of Experimental Algorithmics, vol. 14, 2009.

[8] K. Gutenschwager, A. Radtke, S. Völker, and G. Zeller, "The shortest path - comparison of different approaches and implementations for the automatic routing of vehicles," in Proceedings of the 2012 Winter Simulation Conference, Berlin, Germany, 9–12 December 2012.

[9] M. Piechowiak, P. Zwierzykowski, and M. Stasiak, "Multicast routing algorithm for packet networks with the application of the lagrange relaxation," in *Proceedings of NETWORKS 2010, 14th International Telecommunications Network Strategy and Planning Symposium*, Warsaw, Poland, September 2010, pp. 197–202.

[10] U. Lauther, "An experimental evaluation of point-to-point shortest path calculation on road networks with precalculated edge-flags," in Proceedings of Ninth DIMACS Implementation Challenge, Piscataway, NJ, USA, 13–14 November 2006.

[11] Y. Sharma, S. C. Saini, and M. Bhandhari, "Comparison of Dijkstra's shortest path algorithm with genetic algorithm for static and dynamic routing network," International Journal of Electronics and Computer Science Engineering, vol. 1, no. 2, 2012, pp. 416–425.

[12] S. Pettie, "On the comparison-addition complexity of all-pairs shortest paths," in Proceedings of ISAAC 2002, 13th International Symposium on Algorithms and Computation, Vancouver, BC, Canada, 21–23 November 2002.

[13] J. Hershberger, S. Suri, and A. Bhosle, "On the difficulty of some shortest path problems," ACM Transactions on Algorithms, vol. 3, no. 1, 2007.

[14] R. Cohen and G. Nakibly, "On the computational complexity and effectiveness of n-hub shortest path routing," IEEE/ACM Transactions on Networking, vol. 16, no. 3, 2008, pp. 691–704.

[15] L. Roditty and U. Zwick, "On dynamic shortest paths problems," Algorithmica, vol. 61, no. 2, 2011, pp. 389–401.

[16] B. L. Golden, "Shortest path algorithms: a comparison," Massachusetts Institute of Technology, Operations Research Center, Tech. Rep., October 1975.

[17] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation," Mathematical Programming, vol. 73, no. 2, 1996, pp. 129–174.

[18] P. Biswas, P. K. Mishra, and N. C. Mahanti, "Computational efficiency of optimized shortest path algorithms," International Journal of Computer Science & Applications, vol. 2, no. 2, 2005, pp. 22–37.

[19] C. Demetrescu, S. Emiliozzi, and G. F. Italiano, "Experimental analysis of dynamic all pairs shortest path algorithms," ACM Transactions on Algorithms, vol. 2, no. 4, 2006, pp. 578–601.

[20] D. P. Bertsekas, Network Optimization: Continuos and Discrete Models. Belmont, Massechusetts: Athena Scientific, 1998.

[21] R. K. Ahuja, T. L. Magnati, and J. B. Orlin, Network flows: Theory, algorithms and applications. Englewood Cliffs, N.J.: Prentice-Hall, 1993.

[22] K. Stachowiak, J. Weissenberg, and P. Zwierzykowski, "Lagrangian relaxation in the multicriterial routing," in IEEE AFRICON, Livingstone, Zambia, September 2011, pp. 1–6.

[23] B. Musznicki, M. Tomczak, and P. Zwierzykowski, "Dijkstra-based localized multicast routing in wireless sensor networks," in Proceedings of CSNDSP 2012, 8th IEEE, IET International Symposium on Communication Systems, Networks and Digital Signal Processing, Poznań, Poland, 18–20 July 2012.

[24] F. B. Zhan and C. E. Noon, "A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths," Journal of Geographic Information and Decision Analysis 4, 2000.

[25] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms. MIT Press, 1990.

[26] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik, vol. 1, 1959, pp. 269–271.

[27] S. Saunders, "A comparison of data structures for Dijkstra's single source shortest path algorithm," Honours project, University of Canterbury, Department of Computer Science and Software Engineering, 5 November 1999.

[28] R. B. Dial, "Algorithm 360: shortest-path forest with topological ordering," Communications of the ACM, vol. 12, November 1969, pp. 632–633.

[29] J. Bang-Jensen and G. Gutin, Digraphs: Theory, Algorithms and Applications. London: Springer-Verlag, December 2008.

[30] U. Pape, "Implementation and efficiency of Moore-algorithms for the shortest route problem," Mathematical Programming, vol. 7, no. 1, 1974, pp. 212–222.

[31] D. P. Bertsekas, "A simple and fast label correcting algorithm for shortest paths," Networks, vol. 23, 1993, pp. 703–709.

[32] D. P. Bertsekas, F. Guerriero, and R. Musmanno, "Parallel asynchronous labelcorrecting methods for shortest paths," Journal of Optimization Theory and Applications, vol. 88, February 1996, pp. 297–320.

[33] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," SIAM Journal on Computing, vol. 10, no. 4, 1981, pp. 657–675.

[34] R. W. Floyd, "Algorithm 97: Shortest path," Communications of the ACM, vol. 5, June 1962, p. 345.

[35] S. Warshall, "A theorem on boolean matrices," Journal of the ACM, vol. 9, January 1962, pp. 11–12.

[36] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," Journal of the ACM, vol. 24, January 1977, pp. 1–13.

Rapid Design of Meta Models

Bastian Roth, Matthias Jahn, and Stefan Jablonski
Chair for Databases & Information Systems
University of Bayreuth
Bayreuth, Germany
{bastian.roth, matthias.jahn, stefan.jablonski} @ uni-bayreuth.de

Abstract - **Designing concise meta models manually is a complex task. Hence, newly proposed approaches were developed, which follow the idea of inferring meta models from given model examples. Unlike most approaches in the state of the art, we accept arbitrary model examples independent of a concrete syntax. The contained entity instances may have assigned values to imaginary attributes (i.e., attributes that are not declared yet). Based on these entity instances and the possessed assignments, a meta model is derived in a direct way. However, this meta model is quite bloated with redundant information. To increase its quality, we provide recommendations for applying so-called language patterns like inheritance or enumerations. For this reason, the applicability of those patterns is analyzed concerning the available information gathered from the underlying model examples. In addition to our previously published work, we also support the derivation of meta model changes based on modifications and extensions of the initial example models. Furthermore, change recommendations are provided wherever possible. This new approach for iteratively building, modifying and refining meta models enables users to focus on the real world instances. Consequently, they are not distracted by keeping the meta level in mind and thus are able to design meta models rapidly.**

Keywords - meta model derivation; meta model inference; derivation of meta model changes; refinement of meta models; language patterns

I. INTRODUCTION

In [1], we presented an approach how a concise meta model can be derived from a given set of example models.

The main aim of our work is to support users in defining domain specific languages (DSLs). In general, a DSL consists of three important parts: an abstract syntax, a concrete syntax and a set of semantic rules [2]. The abstract syntax defines language concepts and how they can be linked together. The concrete syntax in turn describes a notation for the visualization of the DSL, whereas the rule set defines the semantics of concepts of the abstract syntax.

Nowadays, developers of a DSL often tend to describe the abstract or concrete syntax with meta models [3]. These meta models are models that specify how their (instance) models are structured. Creating a meta model and hence a DSL is not a trivial task, if it has to be done manually. That is why different methods for developing meta models have been discovered. The most recent approach is the derivation of meta models out of some (possibly merely one) example models.

In the following, when talking from a meta model we always mean the abstract syntax of a DSL. Since it requires a large set of models, we explicitly do not support inference of constraint (e.g., based on OCL). Additionally to that, negative example models are needed as well to avoid overgeneralization [4], [5]. Negative examples are models, which expose an invalid scenario in terms of the intended DSL. In our case, providing such examples is impracticable because it forces the user to pre-think models that are out of the regarding domain's scope.

During the derivation of an abstract syntax, all meta model artefacts are generated automatically and thus, could differ from the user's expectations, especially in terms of quality. In order to achieve a tolerable degree of quality, the user is pointed to parts of the meta model with potential of improvement and also supplied with possible solutions in form of language patterns (e.g., inheritance or enumerations). In contrast to design patterns [6], language patterns are supported by modelling systems themselves and can be utilized in a direct and simple manner.

The development of a meta model is often driven by the evolution in understanding of the domain of interest. Hence, together with the growing knowledge, the meta model often needs to be adapted to fulfil the domain's requirements. Therefore, it is essential that – based on modifications of the example models – changes within the meta model can be derived that define how such a meta model have to or may be adapted to get a concise result again. We call this whole process of incrementally deriving a meta model and providing some recommendations for quality improvements "rapid design of meta models".

After this introduction, some fundamentals are explained, which help understanding the later parts of this paper. Then, an example model is presented that is used for exemplary explanations through the entire paper. Following this, we introduce a method how a meta model can be automatically derived from a given set of such example models. Since this meta model may have some potential for improvements, in the subsequent Section V, two algorithms are presented that detect constellations of meta model elements with the aforesaid improvement potential. Also, each algorithm suggests a suitable solution, which can be applied by the user manually. Beyond tweaking the meta model, example models may be evolved as well or new ones can be added. Thus, in Section VI we describe an approach how freely performed changes at example model side have impacts on an already existing meta model. Afterwards, an overview of some related

work is given. Finally, we look out on future challenges in the field of rapid design of meta models and even whole domain-specific languages.

II. FUNDAMENTALS

In the following three subsections, we explain some fundamentals that act as basis for the subsequently presented rapid design approach.

A. Model Workbench

Model Workbench [7] is a web-based meta modeling platform that targets on supporting developers for creating their own modeling language. In contrast to other tools, it leverages advanced language patterns (e.g., Powertypes [8]) building (meta) models. Its implementation is based on the Orthogonal Classification [9]. Thus, the system provides a Linguistic Meta Model (LMM) [10] and interprets (meta) models at runtime in order to emulate a concrete textual syntax (called Linguistic Meta Language, LML). Together with that, Model Workbench is not limited to any number of meta levels since it is able to manage arbitrary meta model hierarchies. Therefore, it uses Clabjects [11] as a hybrid of a class and an object for representing concepts of a model (the term "concept" means a Clabject throughout the context of Model Workbench). Hence, a concept always has two different facets: a type and an instance facet. As a type (also called a meta concept), a concept defines attributes whereas as an instance (also called an instance concept), a concept contains assignments each of which may be associated with an attribute of an instantiated meta concept.

In general, Model Workbench divides attributes and assignments into two different classes depending on their respective type: literal and referential ones. Literal attributes can have one of the following types: boolean, integer, float, pointer, string or enumeration. In our understanding, enumerations are regarded as literal types, too. That is tolerable because enumerations can also be represented by integers with a highly restricted range of values. Each defined concept, however, may be used as a referential type.

B. Modelling modes

Creating instance models based on a given meta model is a typical use case during modelling. Thereby, the instance models have to satisfy the constraints specified by the meta model. We call this kind of modelling the "stringent (modelling) mode".

By way of contrast, in context of the "free (modelling) mode" the constraints of a possibly available meta model are completely ignored. Accordingly, the LMM as specified in [12] needs to be expanded by schemalessness. Concretely, it means to be able to name an instance concept's type that does not exist (yet). Additionally, it must be possible to create assignments to imaginary attributes. An imaginary attribute is an attribute that is not (yet) declared by a meta concept.

C. Essential assumption on equally named elements

The most important assumption we take is that equally named elements (types of instance concepts on the one hand, assignments and attributes on the other hand) always relate to the same semantic object at domain side. One could imagine a meta model containing two different concepts, each with exactly one string attribute labeled as `owner`. When trying to make this meta model more concise, both concepts are deemed to be candidates for generating a common super concept because of the two equally named attributes.

This assumption is mandatory. Otherwise, neither a meta model can be derived from one or more example models nor elements can be identified that exhibit some potential for improvement. Furthermore, the three inference approaches presented in Section VII follow a comparable principal.

III. EXAMPLE MODEL

Before introducing the different algorithms for deriving a meta model, a linguistic example model (Figure 1) is presented on which we refer to in the following sections. This model is created freely using the LML as concrete syntax (i.e., there is no underlying meta model) and represents a process for planning a conference attendance. It only serves demonstration purposes and hence, it does not lay claim to

```
Start S
  next = Search

ElectronicTask Search
  title = "Conference search"
  next = Request
  duration = 1

PaperTask Request
  title = "Travel request"
  next = Split
  duration = 0.5

And Split
  next = Register, Booking

ElectronicTask Register
  title = "Conference registration"
  duration = 0.3
  next = Join

ElectronicTask Booking
  title = "Book Hotel"
  next = Organize

ElectronicTask Organize
  title = "Schedule journey"
  duration = 0.6
  next = Join

And Join
  next = E

DetachedTask Inform
  title = "Inform about topics"
  duration = 1

Exit E
```

Figure 1. Example model of a process

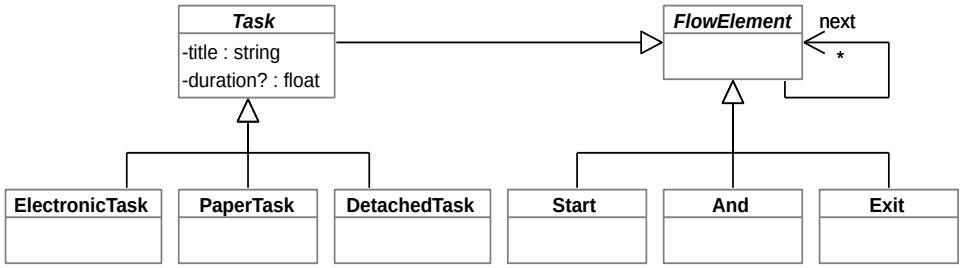


Figure 2. Meta model which matches exmple process model

contentual completeness. Since the according syntax (LML) is quite similar to the one of popular object-oriented programming languages, it is easy to read for software developers and modelers.

The process's flow is as follows. After a suitable conference has been found, an appropriate travel request needs to be submitted. Only then, a hotel may be booked and the journey may be scheduled. In parallel to these two steps, the researcher can also register at the conference. At any time, the scientist may inform herself/himself of the concrete topics covered during the conference.

The successor relationships are reflected in the `next` assignments. Furthermore, each task contains a `title` and can be equipped with a `duration`. The individual steps differ in that they have to be executed electronically (`ElectronicTask`), on paper (`PaperTask`) or besides at an undetermined time (`DetachedTask`). For the mentioned parallel processing, there are the two elements `Split` and `Join` with "and" semantic. The `And` means that all steps of both threads have to be completed before the execution can continue. Finally, there are two further elements, which determine the process's start and end points.

A meta model that matches this example process model is shown in Figure 2. It fulfills important quality criteria specified by Bertoa and Vallecillo in [13]. Looking at `ElectronicTask`, `PaperTask`, `DetachedTask`, `Start`, `And` and `Exit`, it exactly contains those concepts that are used within the example model (completeness). The same is true for the three attributes `title`, `duration` and `next`, which are declared only once and thus, redundancy is avoided. Moreover, because of the base concepts' naming – `Task` and `FlowElement` – their intention is obvious (self-documentation).

The meta model, however, concedes more flexibility as expressed by the underlying example model. For instance, `DetachedTask` is fully unconnected from the whole control flow, but the meta model states that it is a flow element nevertheless. The advantage of this additional flexibility is that when processing detached tasks, in some cases they need not be handled separately. For example, think about a concrete graphical syntax, which should be defined for this meta model. Then, it suffices if one containment mapping is specified for flow elements to lie within a certain process.

Suchlike assumptions concerning a higher degree of flexibility cannot be inferred from the example model. They require a profound knowledge about the particular domain and how according models are processed. Consequently, the meta model cannot be generated automatically as depicted by Figure 2, but it can be approximated to a certain degree. However, for further refinements, recommendations can be provided, which hint the user at sets of model elements with room for improvement. With improvements, we mean language patterns that can be applied to those model elements. Further details about this topic can be found in Section V.

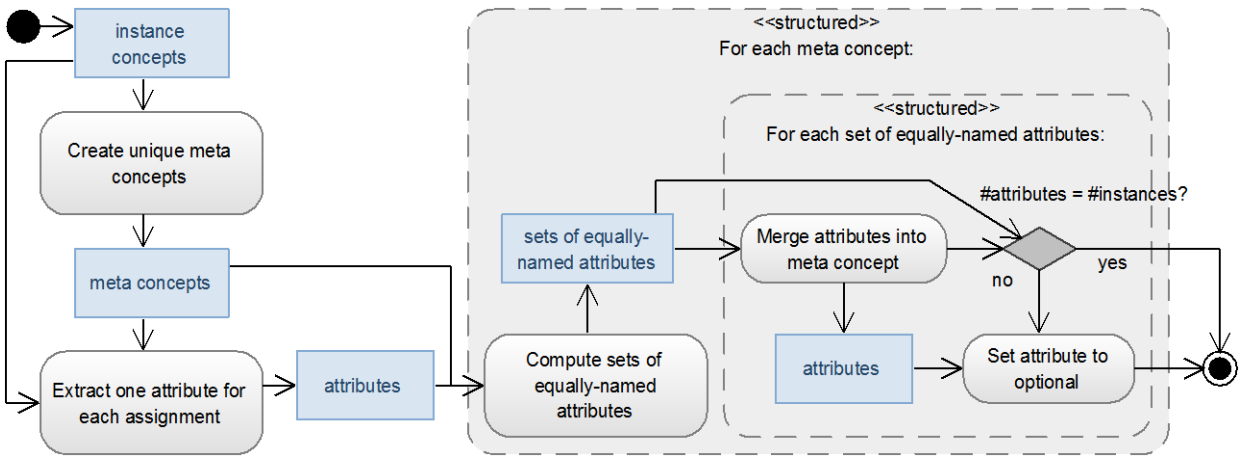


Figure 3. Activity diagram of the initial bottom-up algorithm

TABLE I. DERIVED META CONCEPTS WITH RESPECITVE ATTRIBUTE SETS

Meta concepts	Instance concepts	Attributes	Attribute sets
Start	S	next: ElectronicTask	{ next: ElectronicTask }
ElectronicTask	Search	title: string duration: integer next: PaperTask	{ title: string, title: string, title: string, title: string } { duration: integer, duration: float, duration: float } { next: PaperTask, next: And, next: ElectronicTask, next: And }
	Register	title: string duration: float next: And	
	Booking	title: string next: ElectronicTask	
	Organize	title: string duration: float next: And	
PaperTask	Request	title: string duration: float next: And	{ title: string } { duration: float } { next: And }
And	Split	next[]: ElectronicTask	{ next[]: ElectronicTask, next: Exit }
	Join	next: Exit	
DetachedTask	Inform	title: string duration: integer	{ title: string } { duration: integer }
Exit	E		

IV. DERIVING AN INITIAL META MODEL

In the following, a method (Figure 3) is presented how a meta model can be derived from a set of example models. This method is an extension of the algorithm introduced in [1]. It exhibits some commonalities with the technique described in [14], but goes deeper into potentially occurring problems as well as respective solutions.

The algorithm's input are all instance concepts of the example models. At first, for each unique type name a separate meta concept is generated. Afterwards, for each assignment an associated attribute is created without allocating it to one of the previously generated concepts. Hereby, the upper bound of the attribute's multiplicity can already be determined. It is set to 1 if only one value is assigned, otherwise it is set to *. Identifying the attribute's type is done using regular expressions. For values that have one of the literal types boolean, integer, float, pointer (represented by qualified names) or string, the result is always unambiguous. However, in case only a qualified name is given, a further differentiation is required because the value may either represent another instance concept or an unspecified pointer. If an instance can be found whose name matches the assigned value, then the attribute type is set to the meta-concept of this instance. Otherwise, the attribute is declared as a pointer attribute.

After that, for every meta concept, sets of equally-named attributes are computed that act as base for the actual attribute declaration within the particular meta concept. Which attribute belongs to which meta concept can be ascertained by considering the underlying instance concepts.

For the example shown in Figure 1, TABLE I lists the derived meta concepts as well as the associated sets of equally-named attributes. In respect of a better traceability, the table also contains the underlying instance concepts together with the attributes inferred from the respective assignments. After the computation of the attribute sets, all attributes of each set are merged to one single attribute, which then is added to the particular meta concept. Merging attributes is not a trivial operation. Hence, it is explicated in the next subsection in more detail.

Finally, the last step checks whether the number of attributes of the original set is equal to the number of instances of the particular meta concept. If so, the algorithm terminates. Elsewise, the number of attributes is smaller than the number of instance concepts, which results in denoting the attribute as optional.

A. Merging attributes

Merging attributes is the central activity when deriving a meta model because in doing so, the information and constraints stemming from different attributes are combined to one single attribute. This way, the domain knowledge obtained from the model examples is consolidated by considering the attribute's name, type and multiplicity. Since all attributes of the source set have the same name, it is adopted by the resulting attribute.

TABLE II. SUPPORTED LITERAL DATA TYPES WITH CONVERSION EXAMPLES

	Boolean	Integer	Float	Pointer	String
Boolean	false / true	0 / 1	0 / 1	-	"false" / "true"
Integer	-	3 / -2	3 / -2	-	"3" / "-2"
Float	-	-	0.5 / -3e6	-	"0.5" / "-3e6"
Pointer	-	-		X1 / A.B.c	"X1" / "A.B.c"

1) Merging attributes' multiplicities

During the merging step, for multiplicities merely two values need to be regarded, namely 1 and 1..*. The multiplicity of an initially created attribute is set to 1 if the underlying assignments embraces exactly one value. In case of several values, the multiplicity is set to 1..*. Thus, when merging attributes only the multiplicity's upper bound can be determined. Thereby, the maximum value range is adopted (i.e., 1..* is preferred). Applied to the example from TABLE I, it means that the attribute set `next` of meta concept `And` leads to the multiplicity 1..*.

The lower bound is addressed in a downstream step. It only is set to 0 if there are more instances of the currently processed meta concept than attributes in the momentarily handled attribute set (see the decision node's successor in Figure 3). Then, instances exist, which do not possess an assignment to the current attribute. As an example, take a look at the `duration` attribute of `ElectronicTask` in TABLE I because it merely appears in three out of four instances.

2) Merging attributes' types

Conflating the types of attributes is far more complex. Thereby, literal and referential attributes need to be distinguished.

Literal attributes as defined by the LMM are attributes with one of these types: boolean, integer, float, string or pointer. In case two or more attributes with different literal types are detected, an automatic type conversion takes place, which is similar to the one of dynamic programming languages like JavaScript [15]. Thereby, the type with largest value range is adopted. Consequently, assigned values from a smaller value range have to be converted into the taken data type.

The head row of TABLE II lists all literal data types whereas the value range grows from left to right. Moreover, the table contains some conversion examples (from small to large value ranges). The type pointer occupies a special position in the context of an automatic type conversion since a pointer can solely be transformed into a string. Compatibility to other data types is not given, which results in aborting the derivation algorithm if such a scenario arises.

In TABLE I, a type conversion is required for the `duration` attribute of `ElectronicTask` since it is two times declared as float and one time as integer. Because of a larger value range the resulting type will be float.

If two or more attributes to merge feature different meta concepts as their type, for typing of the consolidated attribute, a common meta concept has to be determined as well. This use case is called Liskov substitution principle and is characteristically for the language pattern "generalization" / "inheritance" [16]. In case a common base concept already is available, it is set as the attribute's type. Otherwise, a suchlike base concept needs to be introduced first.

Referred to TABLE I, this affects the attribute sets `next` of `ElectronicTask` and `And`. As a consequence, for `ElectronicTask`, `PaperTask` and `And` as well as for `ElectronicTask` and `Exit` a base concept has to be created respectively. In Figure 4, these base concepts are represented as `ElectronicTaskOrPaperTaskOrAnd` and `ElectronicTaskOrExit`. The automatic naming happens by means of concatenating the names of the individual source concepts, whereas between two names always "Or" is inserted. Since the diagram shows the initially derived meta model for the example process model from Figure 1 all contained attribute sets are already merged and added to the respective meta concept. The question mark behind a literal attribute's name tells it is as an optional one (e.g., `duration`).

B. Elimination of multiple inheritance

As obvious through Figure 4, the approach presented above may lead to the introduction of multiple inheritance. In several cases this is undesired because it carries some potential risks [17] (e.g., name collision). That is why, an additional operation can be connected in series with the initial derivation process that removes multiple inheritance from the generated meta model. In order to not increasing the meta model's complexity artificially, multiple inheritance is replaced by the language pattern "single inheritance".

The replacement strategy starts by looking for compounds of concepts with multiple inheritance. For each such compound, all base concepts are identified and conflated to one common base concept using evolution techniques as described in [18].

The naming is handled equally to the one from above, i.e., names are concatenated using a connecting "Or". In order to restrict the name's length a bit, common partial strings are only quoted once.

Figure 5 depicts the accordingly modified variant of the meta model from Figure 4. The compound with multiple inheritance initially consists of `ElectronicTask`, `Exit`, `PaperTask`, `And`, `ElectronicTaskOrExit` and

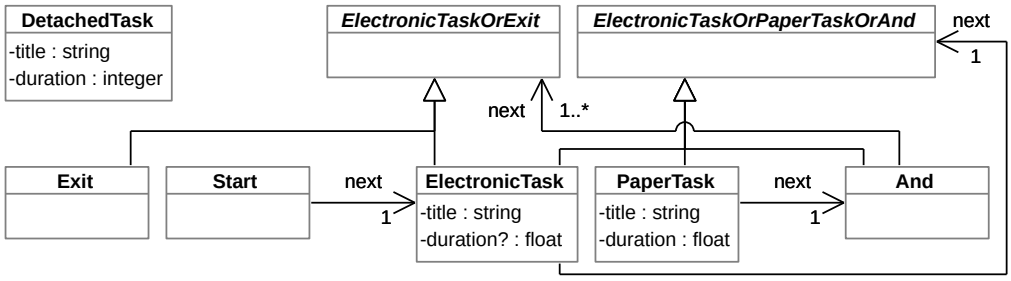


Figure 4. Initially derived meta model with multiple inheritance

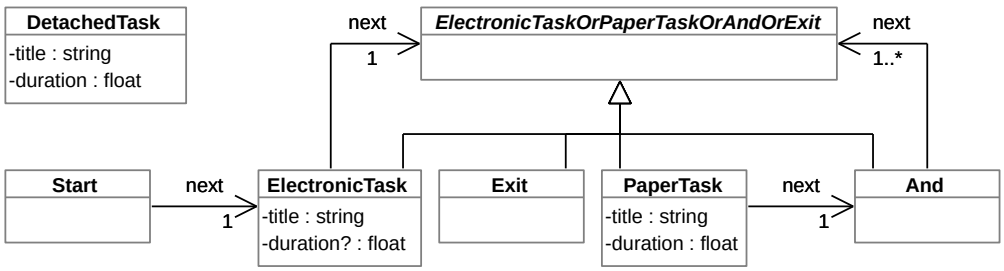


Figure 5. Initially derived meta model with single inheritance

`ElectronicTaskOrPaperTaskOrAnd`. The two latter mentioned meta concepts represent the base types, which are merged into `ElectronicTaskOrPaperTaskOrAndOr-Exit`.

Owing to later manual modifications, base concepts could also contain some attributes that again may result in naming conflicts. Such attributes have to be merged analogous to the method described in Section IV.A. Since this may lead again to more than one base concept per meta concept, the newly introduced multiple inheritance needs to be eliminated in turn. At the latest, this cycle terminates when a global base concept is found, which acts as generalization for all other meta concepts.

As an alternative to the foregoing strategy, instead of conflating the base concepts, the generalization hierarchy can be extended by introducing a super concept for those base concepts. If the concept compound comprises a big number of base concepts, it may result in a complex generalization hierarchy. Because of the large amount of additional concepts, the comprehensibility and thus the meta model's quality suffers [13]. However, the complexity of the meta model is only increased slightly when pursuing the first mentioned solution. Consequently, this one is preferred.

V. META MODEL REFINEMENT

Looking at the initially derived meta model in Figure 5, some parallels to the expected variant in Figure 2 are indeed obvious, but the automatically generated model contains a bunch of redundancies, which impair its comprehensibility. Furthermore, the expected variant comprises already amended domain knowledge, which lacks the generated result. One example is the concept `Task` that specifies as a generalization which kind of information all tasks must/may provide. In this concrete case, it is about a task's title and a time designation how long a `Task` instance will take approximately.

Hence, the requirement arises to rebuild the derived result in a way that it widely corresponds to the expected model. Since inferred meta models can be much bigger than the ones shown in this article, it is desirable to point a modelling expert to constellations of model elements with potential for optimization. This is contrary to the method presented in [1] where optimizations are performed automatically by applying appropriate language patterns. The reason for limiting to recommendations comes from the amount of different possible solutions how a meta model may look like to fit a set of example models.

This becomes clear when looking at Figure 2, Figure 5 and Figure 6, which all are valid according to the example process model and only utilize single inheritance as language pattern. Which one to choose requires additional domain knowledge that is not available to the derivation engine. However, this knowledge is availble to the user and hence, (s)he can decide herself/himself whether to introduce a certain suggested pattern. Also, focusing on this challenge, we develop a framework that provides support for user-oriented meta model evolution [19].

To provide recommendations, we resort to the principle of equally-named attributes explicated in Section II.B. Thereby, in a given meta model, sets of concepts are searched, which declare as many equally-named attributes as possible. Suchlike sets represent candidates for introducing generalizing language patterns. The most widespread generalization pattern is single inheritance. It is addressed in the first subsection.

Another kind of generalization can be achieved using enumerations. An enumeration, however, does not relate to concepts but to literal data types with a limited value range. The basis are again equally-named attributes. This pattern is covered within the second subsection.

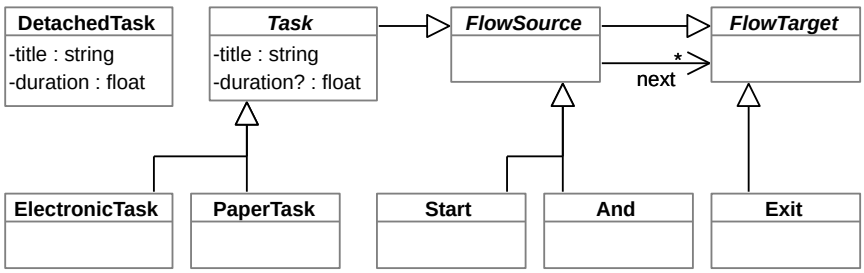


Figure 6. Alternative meta model with single inheritance

A. Single inheritance as refinement recommendation

In order to provide a refinement recommendation for applying single inheritance, attributes need to be searched, which (potentially) have the same meaning. In the following, we call those attributes "corresponding attributes". With regard to Section II.B, two attributes correspond if they coincide by name and kind (i.e., referential or literal). By means of an external configuration it can be specified whether type and multiplicity also have to match such that correspondence is on hand. As opposed to equally-named attributes, corresponding attributes are declared by different meta concepts.

The described correspondence is an equivalence relation, because it is *reflexive* (each attributes corresponds to itself), *symmetric* (if attribute a corresponds to attribute b then b corresponds to a, too), and *transitive* (if attribute a corresponds to attribute b and attribute b corresponds to c then a corresponds to c as well). Consequently, the order of corresponding attributes is irrelevant and thus, it is expedient to represent them in form of sets.

Referred to the meta model in Figure 5, the following attribute sets arise as a result if besides the attribute names no further information is checked on equality:

- `{ DetachedTask.title: string, ElectronicTask.title: string, PaperTask.title: string }`
- `{ DetachedTask.duration: float, ElectronicTask.duration?: boolean, PaperTask.duration: boolean }`
- `{ Start.next: ElectronicTask, ElectronicTask.next[]: ElectronicTask-OrPaperTaskOrAndOrExit, PaperTask.next: And, And.next: ElectronicTaskOrPaperTaskOrAndOrExit }`

In case multiplicity is considered as well, the particular representatives of `ElectronicTask` of the `duration` and `next` attribute sets are dropped. For it, the `duration` is declared as optional while for the other concepts, it is specified as mandatory. The electronic task's `next` attribute, however, permits to assign multiple values whereas the other concepts require exactly one successor to be assigned.

A set of corresponding attributes implies that the declaring concepts of the attributes contained by this set exhibit exactly one correspondence, namely these attributes. In case of the first listed set, the three `title` attributes form a correspondence (communality) of the concepts `DetachedTask`, `ElectronicTask` and `PaperTask`. The same is true for the three `duration` attributes. Consequently, the three concepts `DetachedTask`, `ElectronicTask` and `PaperTask` possess exactly two communalities, which are determined by the two sets of corresponding attributes.

The issue of attribute sets with the same correspondences can be generalized. If two sets of corresponding attributes have the same size and the declaring concepts of the contained

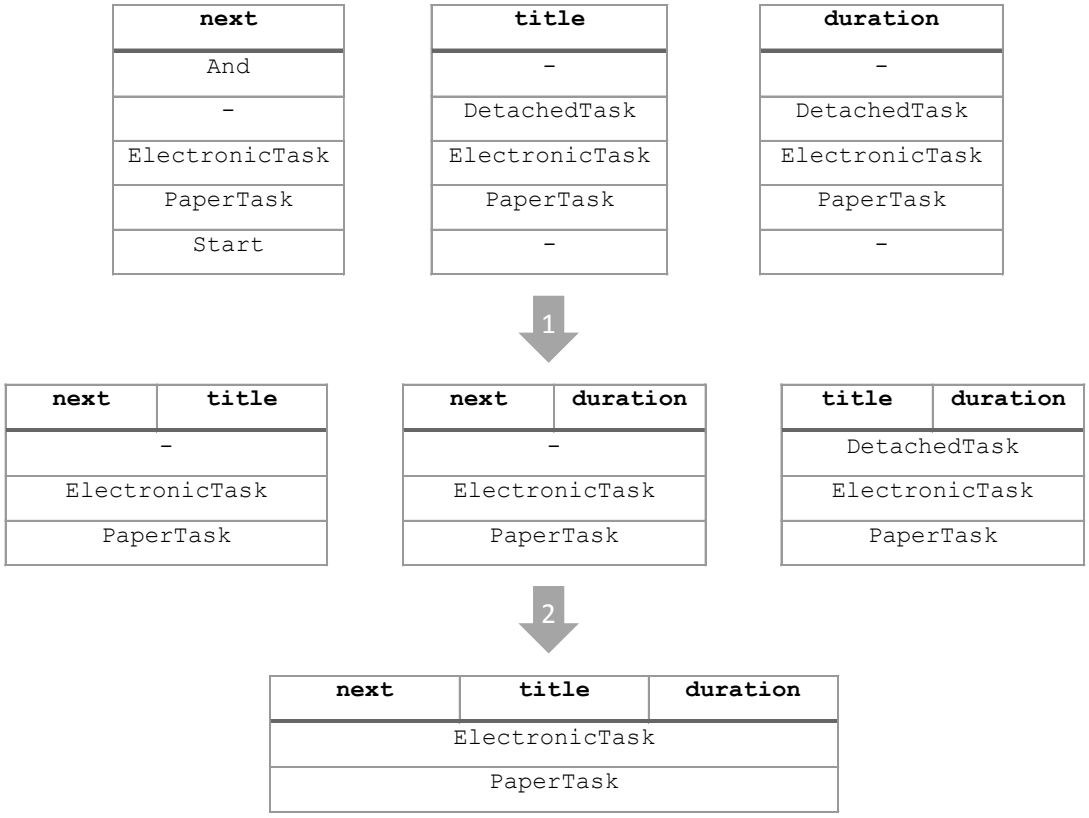


Figure 7. Example for determining dependent sets of corresponding attributes

attributes coincide, then we talk about a dependency between these attribute sets regarding the common parent concepts. Analogous to the corresponding attributes, this dependency relationship constitutes an equivalence relation.

Visualizing this circumstance can be done using tables like in Figure 7. Thereby, for each set of corresponding attributes, the first row contains exactly one entry with the common name of the respectively contained attributes. The rows below list all those concepts that depend on each other based on the attribute sets consolidated within the first row. Cells showing a "-" are included due to purpose of illustration without any contentual meaning. Each one of these tables represents a candidate for applying a generalizing language pattern and thus for refining the meta model in relation to the concepts and attributes listed by the table.

At large, in a meta model many such candidates can be found. Hence, it is important to weight the determined candidates and recommend them to the user ordered by this weight. It is defined by the number of dependent sets of corresponding attributes. As a consequence, a candidate is better than another one if there is a greater number of such attribute sets. In case this number is identical for two attribute sets, the quantity of declaring concepts is considered as secondary factor. It is justifiable because an in fact occurring communality is more probable if two or more concepts overlap in as many points (corresponding attributes) as possible. In Figure 7, it is the case for the table at the bottom. This table states that the concepts `ElectronicTask` and `PaperTask` depend on each other concerning the attribute sets `next`, `title` and `duration`.

The algorithm for determining all refinement candidates is shown in Figure 8 in form of an activity diagram. It starts with looking for corresponding attributes in a given collection of meta concepts. The specific correspondence criteria are predefined externally by means of an configuration.

The found sets of corresponding attributes are then converted into a data structure called "dependency tuple". Its content is exemplarily depicted by the tables in Figure 7. The first entry of such a tuple contains the dependent sets of corresponding attributes and thus, it conforms to the first rows of the example's tables. The second entry comprises those concepts, which declare exactly one attribute of every set of the first entry. These concepts are located in the other rows of the example tables. The three sets of corresponding attributes listed above are equivalent to the first three dependency tuples (represented as tables) in Figure 7.

The next step creates the initial dependency tuples and puts it at the beginning of the results list. The results list contains the refinement candidates, which are identified during the execution of the algorithm. The tuples are ordered descending by the quantity of included concepts. Accordingly, the first entry is always the candidate with the greatest probability in terms of an in fact occurring communality within the real world.

If there are at least two dependency tuples, they are combinated in pairs with formation of intersecting the declaring concepts. Thereby, dependency tuples are created only for such intersections, which contain at least two concepts since elsewise no dependency exists. This combination step is repeated as long as one tuple is left at a max. After that, the algorithm terminates and returns a list of refinement candidates ordered by the weight described above.

Applied to the example depicted by Figure 7, the results list looks as follows (for reason of clarity, solely the names of the corresponding attributes are specified):

```
({next, title, duration},
 {title, duration},
 {next, title},
 {next, duration},
 {next},
 {title},
 {duration})
```

At first place, it recommends the user to introduce a common base concept for `PaperTask` and `ElectronicTask`, which declares the three corresponding attributes `next`, `title` and `duration`. If (s)he does not want to do that (s)he can look at the next candidate. Based on the attributes `title` and `duration`, it recommends to introduce a base concept for `DetachedTask`, `PaperTask` and `ElectronicTask`. This can be continued until the last dependency tuple is arrived that only rests on `duration`.

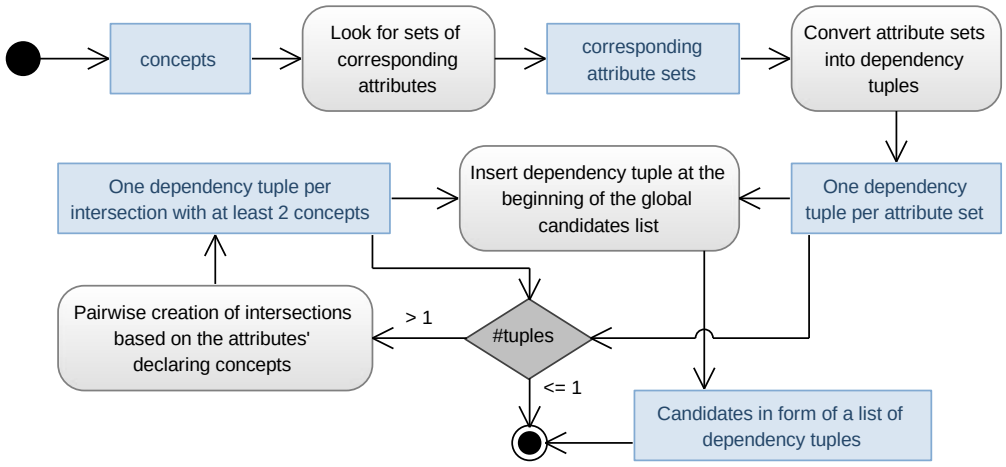


Figure 8. Algorithm for the computation of candidates to apply single inheritance

```
Job J1
  phase = PRE

Job J2
  phase = POST

Job J3
  phase = PRE

Job J4
  phase = DEFAULT

Job J5
  phase = DEFAULT

Job J6
  phase = PRE
```

Figure 9. Example model that induces a recommendation for introducing an enumeration

B. Enumeration as refinement recommendation

An enumeration represents a data type with a strongly limited value range [20]. In general, it only consists of a few literals, which come into question as values for assignments. Therefore, recommending the introduction of an enumeration as data type is merely reasonable for corresponding attributes whose assignments exhibit repeatedly the same values. Owing to the equal lexical structure of pointers and enumeration literals, an enumeration can be only intended for pointer attributes by users. Consequently, in the current context merely two attributes may correspond to each other if they feature the same name and are of type pointer.

Additionally, the number of the different values should be stinted. However, a fix definition of where the border of "stinted" is exceeded cannot be given because this depends on the particular operational scenarios as well as the user's preferences. Instead, the analysis's focus lies on the repeated assignment of the same pointer values to corresponding attributes. Hence, it will be recommended to introduce an enumeration if at least two different values are repeatedly assigned to the same set of corresponding attributes. An example for a valid scenario is shown by Figure 9. It represents a model with six instances that all contain an assignment to the imaginary attribute `phase`. The derived meta model only consists of the concept `Job`, which manifests the aforementioned attribute `phase`. Since it is a pointer attribute and the literal `PRE` as well as the literal `DEFAULT` are used by at least two associated assignments (namely `J1`, `J3`, `J5` and `J4`, `J5`, respectively), a hint is generated that suggests to introduce an enumeration.

VI. Deriving Meta Model Changes

When deriving meta model changes, the fundamental principle is to keep those changes to a minimum. Thus, the existing meta model only gets adapted insofar that modified or newly added example models become valid. This is necessary because users are allowed to commute meta models arbitrarily. In case an existing meta model is always discarded and a complete re-generation takes place, all manually performed modifications would be lost. Which modifications are performed at the meta model automatically during the repeated derivation is explicated in the first subsection.

In the second subsection, we seize the idea of recommendations. Primarily, these recommendations can be seen as counterparts to the explicit and implicit impacts on the meta model presented in Section IV and Section V.

A. Required changes

In order to ensure the conformity of the example models with regards to the meta model, in any case those artefacts of the models need to be extracted that conflict with the meta model. Potential for conflicts is carried by the LMM's parts, which are extended about schemalessness (Section II.B). On the one hand, these are type names of concepts and on the other hand these are names of assignments.

If free modelling mode is enabled, the user may equip new instance concepts with a type name of a not yet available type (meta concept). Suchlike instances are handled the same way as during the initial inference of a meta model (Section IV). A user may also change a type name of an existing instance concept, which is already linked with a meta concept, such that it does not fit with any other available meta concept. Then, this concept is considered as new, too. Furthermore, potentially present assignments are broken away from their underlying attributes. Afterwards, processing can continue in the same way as with completely new instances.

The free mode enabled, new dynamic assignments (i.e., assignments without an underlying attribute) can be created inside of instance concepts, which already have an associated meta concept. For every suchlike assignment an appropriate attribute is generated, but without putting it into a meta concept. After that, per meta concept sets of equally-named attributes are determined. Each of these sets is merged to one attribute and added to a particular meta concept, according to the method described in Section IV.A. Thereby, an existing meta concept is expanded by an attribute that matches one or more dynamic assignments.

Furthermore, assignments with an underlying attribute may feature arbitrary values on the right side provided that the respective intention (referential or literal) is not violated. Assuming that there is an integer attribute with name "height", then assignments may be of any other literal type in free mode. For instance, a meaningful value would be 3.5 although it is outside the value range of integers. Deriving the according meta model changes would convert the "height" attribute's type to float. Here again, strategies are reused, which have been introduced for inferring an initial meta model (Section IV). In case of an underlying literal attribute, a type conversion occurs towards a larger value range (examples are depicted by TABLE II). For referential attributes, however, a common base concept is required, which has to be created if not yet existent.

In addition, enumeration attributes need to be handled separately. Valid values are basically pointers that do not reference another concept. If values are specified without a suitable enumeration literal, an according literal is generated and added to the enumeration. Beyond pointers, string values

can also be assigned to enumeration attributes while free modelling. This, however, leads to converting the underlying attribute to a string attribute. Besides, all enumeration literals are converted to strings as well. All other data types are not permitted and will result in aborting the derivation process in case they are used.

The presented five cases encompass all possible modification kinds of instance models that require a subsequent adaption of the underlying meta model to achieve validity when modelling stringently.

B. Change recommendations

The different types of change requirements can be divided into three categories. In the first category there are all recommendations that affect the value ranges of attributes. The second category encompasses recommendations to delete certain concepts of the meta model. The third one contains those recommendations that refer to a removal of language patterns. Therewith this class stands inverse to the suggestions presented in Section V.

1) Narrowing of attribute contraints

During the derivation of attributes all restriction in the model are softened. This is desirable for reasons of manual adaption. Instead, under certain circumstances narrowing the attribute's multiplicity or type can be recommended. For recommending the narrowing of an attribute's multiplicity, the minimum and maximum have to be handle separately. If all instances that can define an assignment do have such an assignment, a change of the minimum from 0 to 1 is suggested. Furthermore, narrowing the maximum of the multiplicity can be useful if the current value is * and all assignments are just single valued.

Dealing with the attribute's type requires again to distinguish between literal and referential attributes. A literal attribute can be checked whether all according assignments have a lower range than previously defined (TABLE II). In this case a replacement of the old type with the new literal type can be recommended. One could imagine that an attribute's type is float and all assigned values are within the integer rage. Hence, a change of the attribute type may be expedient.

Referential attributes can be handled in a similar way. However, they are tested whether a generalization of their type can be replaced by a specialization of it. Thereby, all assigned values have to be checked again. An example would be an attribute of type `ProcessOrAnd`. This type has been chosen because until now only processes and AND gateways have been assigned. During the next derivation of changes it is detected that only instances of `Process` were used as values. According to that, changing the attribute's type to `Process` is recommended.

2) Concept removal

Based on changed instance models, a sure decision whether a concept is not needed any longer and thus can be deleted is hard to make. Every meta concept may be used in a model repository out of the current scope or needed within a code-generation step. That is why deleting a meta concept is not done automatically but could be done by a modelling expert who is supplied with a recommendation of an according deletion operation. A typical representative would be a non-abstract meta concept, which is not instantiated. Such a concept is a candidate for removal.

3) Revocation of single inheritance

As stated above, the next case can be seen as opposite to the introduction of language patterns explicated in Section V. However, it claims for removing meta concepts, which again may lead to invalid external references. Hence, a model expert has to decide whether (s)he wants to adapt the model or not. If an abstract concept has exactly one specialization this concept is often obsolete. Thus, every concept that fulfils this constraint is a candidate for inlining into its specialization. One could imagine that the concept `PaperTask` (Figure 6) was removed manually. After that, a hint will be generated recommending the move of the two attributes `title` and

```
Start S
  next = Search

ElectronicTask Search
  title = "Conference search"
  next = Request
  duration = 1

PaperTask Request
  title = "Travel request"
  next = Split
  duration = "0.5"
  waitForReturn = true

And Split
  next = Register, Booking

ElectronicTask Register
  title = "Conference registration"
  duration = 0.3
  next = Join

ElectronicTask Booking
  title = "Book Hotel"
  next = Organize

ElectronicTask Organize
  title = "Schedule journey"
  duration = 0.6
  next = Join

And Join
  next = E

FreeTask Inform
  title = "Inform about topics"
  duration = 1

Exit E
```

Figure 10. Manually modified example model

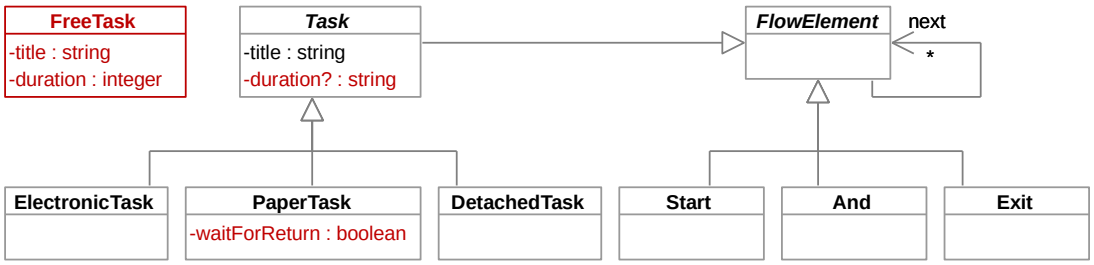


Figure 11. Automatically adapted meta model according to the modified example model

`duration` from `Task` to the specialization `Electronic-Task` and replacing the concept `Task` by `ElectronicTask` afterwards.

C. Example

The example model depicted by Figure 10 largely represents the same process as shown by Figure 1. Both models only vary on three user-performed changes, which are highlighted in Figure 10 using red color. For the original example, a meta model has already been generated. Also, it has been adapted by the user so that it corresponds to the variant from Figure 2. Now, this meta model constitutes the foundation for deriving changes based on the modifications of the example model described below.

The first change concerns the `Request` concept's `duration` assignment that is already bound to a float attribute. The assignment of the floating point number `0.5` is replaced by the string `"0.5"` (fourth issue in Section VI.A). During the incremental meta model derivation, the type of the existing literal attribute is widened to string (concept `Task` in Figure 11) and all previously assigned values to this attribute are converted to their string representation. For instance, the value `1` from `Search` concept's `duration` assignment becomes `"1"`.

Directly below the modified duration assignment, a new assignment (`waitForReturn`) has been added without an underlying attribute (third issue in Section VI.A). Since it affects the one and only instance of `PaperTask`, this meta concept is simply extended by an appropriate Boolean attribute.

The third manipulation of the example model affects the `Inform` concept. Its type has been changed from `DetachedTask` to `FreeTask`, whereas no corresponding meta concept exists for the latter. According to the second issue in Section VI.A, a new meta concept called "FreeTask" is induced that also receives two attributes `title` and `duration`. As a result, there is no more concept, which instantiates `DetachedTask`. For that reason, the system suggests to the user to delete this concept (as per Section VI.B.2).

VII. Related Work

As mentioned in the introduction, deriving a meta model from a set of model examples is not a totally new approach. Depending on their purpose, the available related work can be classified into two categories: meta model reconstruction and meta model creation.

Meta model reconstruction stems from the field of grammar reconstruction and grammatical inference [21]. Thereby, many textual sentences (ideally positive and negative samples) are analyzed to infer a grammar [22].

In current research, the Metamodel Recovery System (MARS) is one prominent representative for meta model reconstruction [5], [23]. It receives a set of model samples and transforms them to a representation that can be used by a grammar inference engine. The output of this engine (a grammar) is then converted back to an equivalent meta model. As the title suggests, MARS focuses on the recovery of meta models (e.g., if a meta model got lost). To obtain a meta model, which corresponds as much as possible to the original one, a large number of positive model samples is required. Otherwise, the resulting meta model is strongly restricted in its capabilities. Since we mostly receive only one or at least a small set of model examples this approach is not practicable for us.

Up to our knowledge, there are three research groups that generate a meta model by deriving it from very few model examples. BitKit as one representative has a rather different intention [24]. Its authors aim at supporting the pre-requirements analysis of software products by allowing to model in a freeform way just like with general purpose office tools. The resulting meta model is merely a means to an end. Primarily, BitKit semantically combines equally looking elements by deriving a common associated entity. After a meta model is inferred and, for instance, the color of such an element is changed the color of every other (equally looking) element is adapted accordingly. Due to the office tool intention of BitKit, the generated meta model is not intended to be processed in any further way. Consequently, its quality is not considered as well.

Another approach is proposed in [25]. Like BitKit, it is also restricted to graphical DSLs. Nevertheless, we adopt their general idea for applying patterns when inferring a meta model. That meta model (which represents the abstract syntax as stated by the author) highly corresponds to the concrete syntax as well. This correspondence is obvious when investigating another publication of Cho and Gray. In [26], they introduce some design patterns well suited for meta models. However, the presented patterns are very specific for graphical DSLs and hence not universally valid. That can be verified when comparing these patterns to the meta models for visual languages defined in [27]. In contrast to our approach, they directly apply design patterns wherever possible. Owing to the visual information, they can resort to additional domain

knowledge, which we do not have on hand. However, our recommendation framework can also be applied to their meta models and hint to artefacts of these meta models with potential for further refinement.

In parallel to our research, a similar approach has been published in [14]. They infer a meta model from example models, which are specified using a predefined textual concrete syntax. From their approach, we adopted the idea of providing recommendations such that a meta model's quality can be increased. Since [14] is rather an overview paper, the authors do not provide detailed solutions how detection of recommendations works. In this article, we minimized that gap and presented some concrete methods how constellations of meta model elements with potential for refinement can be identified.

VIII. OUTLOOK

The presented rapid design approach works well for meta models, which are formulated using a linguistic meta language as concrete syntax. For entire DSLs, further effort is necessary since each DSL features its own concrete syntax whose specification process should also follow the proposed rapid design principle. For sketching textual concrete syntaxes, we already published a method in [28].

Our next step is to combine the meta model derivation approach presented in the current paper with the construction of custom concrete syntaxes. Beyond textual syntaxes, we also contemplate to support graphical DSLs.

To conclude, the overall goal is developing a system, which fosters the rapid design and usage of all artefacts DSLs consist of. This means that the intended system provides a seamless integration of free and stringent modelling when working with meta models and even entire DSLs.

ACKNOWLEDGMENT

This article was authored in the context of the project "Kompetenzzentrum für praktisches Prozess- und Qualitätsmanagement" (KpPQ) funded by "Europäischer Fonds für regionale Entwicklung" (EFRE). So, we thank this institution, which has kindly facilitated our work.

REFERENCES

[1] B. Roth, M. Jahn, and S. Jablonski, "A method for directly deriving a concise meta model from example models," in *Proceedings of the 5th International Conferences on Pervasive Patterns and Applications*, 2013, vol. 5, no. 1, pp. 52–58.

[2] T. Clark, P. Sammut, and J. Willans, *Applied Metamodelling: A Foundation for Language Driven Development*. CETEVA, 2008, p. 227.

[3] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*, 1st ed. John Wiley & Sons, 2008, p. 444.

[4] E. M. Gold, "Language identification in the limit," *Inf. Control*, vol. 10, no. 5, pp. 447–474, 1967.

[5] F. Javed, M. Mernik, J. Gray, and B. R. Bryant, "MARS: a metamodel recovery system using grammar inference," *Inf. Softw. Technol.*, vol. 50, no. 9–10, pp. 948–968, 2008.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[7] B. Roth and M. Jahn, "Model Workbench," 2013. [Online]. Available: http://www.ai4.uni-bayreuth.de/de/research/projects/003_ModelWorkbench/index.html. [Accessed: 28-Jan-2013].

[8] J. Odell, *Advanced object-oriented analysis and design using UML*. Cambridge University Press, 1998.

[9] C. Atkinson and T. Kühne, "Concepts for comparing modeling tool architectures," in *Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems*, 2005, pp. 398–413.

[10] B. Volz and S. Jablonski, "Towards an open meta modeling environment," in *Proceedings of the 10th Workshop on Domain-Specific Modeling*, 2010.

[11] C. Atkinson and T. Kühne, "Meta-level independent modelling," in *Proceedings of the International Workshop on Model Engineering at 14th European Conference on Object-Oriented Programming*, 2000, pp. 12–16.

[12] B. Volz, "Werkzeugunterstützung für methodenneutrale Metamodellierung," University of Bayreuth, PhD thesis, 2011.

[13] M. F. Bertoa and A. Vallecillo, "Quality attributes for software metamodels," in *Proceedings of the 13th TOOLS Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, 2010.

[14] J. Sánchez-Cuadrado, J. De Lara, and E. Guerra, "Bottom-up meta-modelling: an interactive approach," in *Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems*, 2012, pp. 3–19.

[15] D. Flanagan, *JavaScript: The Definitive Guide*, 6th ed. Sebastopol, CA: O'Reilly Media, Inc., 2011, p. 1078.

[16] B. Liskov, "Data abstraction and hierarchy," *ACM SIGPLAN Not.*, vol. 23, no. 5, pp. 17–34, 1988.

[17] G. Singh, "Single versus multiple inheritance in object oriented programming," *ACM SIGPLAN OOPS Messenger*, vol. 6, no. 1, pp. 30–39, 1994.

[18] M. Herrmannsdoerfer, S. D. Vermolen, and G. Wachsmuth, "An extensive catalog of operators for the coupled evolution of metamodels and models," in *Proceedings of the 3rd International Conference on Software Lanugage Engineering*, 2010, pp. 163–182.

[19] M. Jahn, B. Roth, and S. Jablonski, "Remodeling to powertype pattern," in *Proceedings of PATTERNS 2013*, 2013, pp. 59–65.

[20] J. Bloch, *Effective Java*, 2nd ed. Upper Saddle River, New Jersey: Addison-Wesley Longman, 2008, p. 384.

[21] M. Mernik, D. Hrncic, B. R. Bryant, A. P. Sprague, J. Gray, Q. Liu, and F. Javed, "Grammar inference algorithms and applications in software engineering," in *22th International Symposium on Information, Communication and Automation Technologies*, 2009, pp. 1–7.

[22] F. King-Sun and T. L. Booth, "Grammatical inference: introduction and survey - part I," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 3, pp. 343–359, Mar. 1986.

[23] Q. Liu, B. R. Bryant, and M. Mernik, "Metamodel recovery from multi-tiered domains using extended MARS," in *Proceedings of the 34th IEEE Annual Computer Software and Applications Conference*, 2010, pp. 279–288.

[24] M. Desmond, H. Ossher, I. Simmonds, D. Amid, A. Anaby-Tavor, M. Callery, and S. Krasikov, "Towards smart office tools," in *SPLASH 2010 Workshop on Flexible Modeling Tools*, 2010.

[25] H. Cho, J. Gray, and E. Syriani, "Creating visual domain-specific modeling languages from end-user demonstration," in *ICSE Workshop on Modeling in Software Engineering*, 2012, pp. 22–28.

[26] H. Cho and J. Gray, "Design patterns for metamodels," in *Proceedings of the SPLASH '11 Workshops*, 2011, pp. 25–32.

[27] P. Bottoni and A. Grau, "A suite of metamodels as a basis for a classification of visual languages," in *Symposium on Visual Languages and Human Centric Computing*, 2004, pp. 83–90.

[28] B. Roth, M. Jahn, and S. Jablonski, "On the way of bottom-up designing textual domain-specific modelling languages," in *Proceedings of the 2013 ACM Workshop on Domain-Specific Modeling*, 2013, pp. 51–55.

Detecting Software Usability Deficiencies Through Pinpoint Analysis

Dan E. Tamir, Divya K. V. Dasari
Oleg V. Komogortsev, Gregory R. LaKomski
Department of Computer Science
Texas State University
San Marcos, Texas USA
{dt19, dd1290, ok11, gl1082}@txstate.edu

Carl J. Mueller
Department of Computer Information Systems
Texas A&M University Central Texas
Killeen, Texas, USA
muellercj@ct.tamus.edu

Abstract— **The effort-based model of usability is used for evaluating user interface (UI), development of usable software, and pinpointing software usability defects. In this context, the term pinpoint analysis refers to identifying and locating software usability deficiencies and correlating these deficiencies with the UI software code. For example, often, when users are in a state of confusion and not sure how to proceed using the software, they tend to gaze around the screen trying to find the best way to complete a task. This behavior is referred to as excessive effort. In this paper, the underlying theory of effort-based usability evaluation along with pattern recognition techniques are used to produce an innovative framework for the objective of identifying usability deficiencies in software. Pattern recognition techniques and methods are applied to data gathered throughout user interaction with software in an attempt to identify excessive effort segments via automatic classification of segments of video files containing eye-tracking results. The video files are automatically divided into segments using event-based segmentation, where a segment is the time between two consecutive keyboard/mouse clicks. Subsequently, data reduction programs are run on the segments for generating feature vectors. Several different classification procedures are applied to the features in order to automatically classify each segment into excessive and non-excessive effort segments. This allows developers to focus on the excessive effort segments and further analyze usability deficiencies in these segments. To verify the results of the pattern recognition procedures, the video is manually classified into excessive and non-excessive segments and the results of automatic and manual classification are compared. The paper details the theory of effort-based pinpoint analysis and reports on experiments performed to evaluate the utility of this theory. Experiment results show more than 40% reduction in time for usability testing.**

Keywords- Software Development; Software Usability; Human Computer Interaction; Pinpoint Analysis; Pattern Recognition; Clustering

I. INTRODUCTION

One of the primary goals of software is to simplify various tasks and enable users to accomplish tasks with ease and efficiency. Numerous fields have recently witnessed an increase in software development and deployment. Nevertheless, feedback from software applications end-users consistently shows that software is at times non-productive, confusing, counter-intuitive, and unsatisfactory [1]-[5]. Clearly, if the users experience problems or difficulty, it is highly unlikely that they will use that software again. Hence, it is very important for software engineers to place significant emphasis on usability evaluation and testing in order to eliminate user complaints and provide the user with a good experience [1]-[5].

Software engineers use a wide variety of tools, such as prototyping, inspection, usability testing, and iterative processes to ensure that the software they produce is usable [1]-[5]. Still, these tools may not address the usability problem efficiently, resulting in a low ranking on usability for several systems [1][5]. The classical methods used in identifying usability techniques have not proven to be very proficient in accurately locating the specific segment of code that could be leading to the usability problems. Without proper data to understand which part of code is faulty, developers would have a hard time identifying and fixing code that leads to usability issues.

The usability testing process involves observing users engaged with a software application and obtaining a set of characteristics of the user experience. This methodology requires an expert to construct, conduct, and assess the tests; as well as devoted laboratory facilities and several users that participate in the tests. Despite all of these efforts, generally usability testing indicates that a problem exists but does not identify the root cause for the problem [1][5]. This makes usability-testing time consuming, expensive, and frustrating for both developers and managers. Hence, it is often ignored.

Most of the tools used to evaluate the usability of a software application use 'time to complete a task', referred to as *time on task* (*ToT*), as a measure for evaluating usability [1]-[12]. This approach of giving high weight to *ToT* may not produce accurate results when factors like system performance, network delays, and interface design, which are difficult to avoid, play a role. An alternate approach is to measure usability in terms of user-effort, which eliminates some of the system issues mentioned earlier, allowing software engineers to focus on the interface design [1][5].

The *Effort-Based Usability Model* of [1][5][6][9]-[12] can be used for setting usability requirements, evaluating

user interface (UI), development of usable software, and pinpointing software usability defects. It is developed using the principle that usability is an inverse function of effort. The model is used for comparison of different implementations of the same application. The results of several experiments conducted on the effort-based model show a strong relationship between effort and usability [1][5][6][9]-[12].

The underlying theory of the *Effort-based Model* is used to produce a framework for identifying usability deficiencies in the software. Accurately locating software usability issues and correlating these issues with UI software code is referred to as Pinpoint Analysis [1][9]-[12]. For example, users who are in a state of confusion, and users that are not sure how to use the software, tend to look around the screen to determine the best way to accomplish a task. This behavior, which can be observed by eye tracking [13][14], is referred to as an excessive search or as excessive effort [1][5][9]-[12]. Identifying and pinpointing excessive effort behavior helps UI designers to rectify numerous usability related issues.

The hypothesis of this research is that it is feasible to devise a framework that can automatically identify *excessive effort segments* by applying pattern recognition techniques, such as K-means clustering algorithm, thresholding, principal component analysis (PCA), and feature selection [15]-[17]. Usability experts can further inspect the *excessive effort segments*. Hence, the automatic part can save experts' time and increase experts' accuracy.

To validate the hypothesis, this research, attempts to evaluate the utility of pinpointing UI deficiencies using pattern recognition techniques for identifying excessive effort in temporal segments of user software interaction. The process of segmentation of user's software interaction session and linking segments is done automatically using the time slice between two consecutive mouse/keyboard clicks. Automatic identification of segments with *excessive effort* behavior reduces the time required for UI designers to analyze and rearrange the interface at the pinpointed time snapshot. The last phase of the pinpoint process involves an expert evaluating *excessive effort segments*. Nevertheless, the process of identifying these segments is automatic and non-supervised.

The main contribution of this work is the development of a new methodology for assessing the usability of software. This methodology helps optimizing the time spent on usability testing while also more accurately identifying specific segments of code that could be leading to the usability issues.

Several experiments were conducted to validate the hypothesis and evaluate the new framework for pinpointing software usability issues. Experiment results show more than 40% reduction in time for usability testing.

The rest of this paper, which is an expanded version of [1], is organized as follows. Section II contains background information. Section III summarizes related work. Section IV presents the experimental setup. Section V details the experiments performed. Section VI presents experiment results and Section VII contains results evaluation. Section VIII concludes the paper with a summary of findings and proposals for further research.

II. BACKGROUND

A. Software Usability

According to the International Organization for Standardization/International Electro-technical Commission (ISO/IEC) 9126 standard, software usability is: "The capability of a software product to be understood, learned, used, and be attractive to the user when used under specified conditions" [8][9]. The standard lists several characteristics that play an important role in defining software usability: understandability, learnability, operability, and attractiveness [8][9]. The effort-based theory focuses on the first three characteristics.

Understandability helps determine how easy it is to comprehend and use the software. It is the ability of a user to understand the capabilities of the software and its suitability to accomplish specific goals. Learnability indicates the ease with which a user learns to use specific software. Operability is the capability of a user to use the software to accomplish a specific goal. Generally, the end-goal of a software application is to enable performing a task efficiently. As such, operability plays an important role in usability. Attractiveness relates to the requirement that the end-user's experience is pleasant and rewarding. The next section discusses several classical usability evaluation methodologies.

B. Classical Methods for Measuring Usability

The classical usability measurements methods are broadly classified into methods that make use of data gathered from users and methods that rely on usability experts. There are usability evaluation methods that apply to all stages of design and development, from product definition to final design modifications. Usability methods are further classified into *cognitive modeling methods*, *inspection methods*, *inquiry methods*, *prototyping methods*, and *testing methods*.

Cognitive models are based on psychological principles and experimental studies to determine times for cognitive processing and motor movements. They are used to improve user interfaces or predict problem areas during the design process. In general, cognitive modeling involves creating a computational model to estimate how long it takes for users to perform a given task [1]-[5]. It involves one or more evaluators inspecting a user interface by going through a set of tasks by which understandability and ease of learning are evaluated. The user interface is often presented in the form of a paper mock-up or a working prototype; but it might be a fully developed interface.

The inspection method involves cognition with emphasis on a hands-on approach. Under the inspection

method, experimenters observe users while they are using the software. The testing and evaluation of programs is done by an expert reviewer. This provides quantitative data, as tasks can be timed and recorded. In addition to quantitative data, qualitative user experience data are collected. Although some of the data collected is qualitative and potentially subjective, it provides valuable information [2]-[4].

Experts obtain information about users' likes, dislikes, needs, and understanding of the system by talking to them, observing them using the system, and through verbal or written questionnaires. Since this information is collected by inquiring and getting direct feedback from users, this model is called the inquiry method [1]-[5].

While the above methods focus on usability testing at an advanced stage in the development, the prototyping method tries to improve usability by refining and providing feedback as the software is being developed. Rapid prototyping is a method used in early stages of development to validate and refine the usability of a system. It is used to quickly and efficiently evaluate user-interface designs without the need for an expensive working model. This helps to remove the developer's resistance to design changes since it is conducted before any actual programming begins. Testing methods provide usability evaluation through testing of users for the most quantitative data. User interaction sessions are observed via two way mirrors or recorded on video that provides task completion time and allows for evaluation of user attitudes [1]-[5].

C. The Effort-based Usability Model

Several studies indicate that many system users associate the physical and mental effort required for accomplishing tasks with the usability of the software [1][5][6][9]-[12]. The *effort-based usability model* for software usability stems from the notion that the usability is an inverse function of effort. For example, an eye tracking device can be used to measure the effort expanded by the user in navigating through the user interface of software. According to the effort-based usability theory, the eye effort is inversely proportional to the operability of the software.

Physical and mental effort are obtained and inferred from logging user activity such as manual activities in the form of mouse movements and eye activities. For this model, *E* denotes the total effort required to complete a task with computer software and is defined as:

$$E = \begin{pmatrix} E_{mental} \\ E_{physical} \end{pmatrix}$$

$$E_{mental} = \begin{pmatrix} E_{eye\_mental} \\ E_{other\_mental} \end{pmatrix}$$

$$E_{physical} = \begin{pmatrix} E_{manual\_physical} \\ E_{eye\_physical} \\ E_{other\_physical} \end{pmatrix}$$

E_{eye_mental} denotes the amount of mental effort to complete the task measured by eye related metrics.

E_{other_mental} denotes the amount of mental effort measured by other metrics.

$E_{physical}$ denotes the amount of physical effort needed to complete the task.

$E_{manual\_physical}$ denotes the amount of manual effort required to complete the task. Manual effort includes, but is not limited to, the movement of fingers, hands, arms, etc.

$E_{eye\_physical}$ denotes the amount of physical effort *invested* in the process of interaction, measured by eye movement related metrics [13].

$E_{other\_physical}$ denotes the amount of physical effort measured by other metrics.

Consequently, the effort required to complete tasks is associated with software usability [1][5][9]-[12]. Physical effort includes manual effort and physical eye effort. In the case of interactive computer tasks, it is possible to calculate effort as a linear combination or a weighted sum of metrics such as the number of mouse clicks, number of keyboard clicks, *average eye path traversed* as well as other eye activity measures, and *mouse path traversed* [1][9]-[12].

Mental effort is essentially the amount of brain activity required to complete a task. To some extent, brain activity, related to a task, can be approximated by processing eye movement data recorded by an eye tracker [1][5][9]-[14]. Eye trackers acquire eye position data and enable classifying the data into several eye movement types useful for eye related effort assessment. The main types of eye movements are [13][14]:

1) *Fixation* – eye movement that keeps an eye gaze stable with regard to a stationary target providing visual pictures with high acuity. Fixations might be a result of "interest" or a result of confusion. In the context of task completion, fixations are generally correlated to confusion.

2) *Saccade* – rapid eye movement from one fixation point to another.

3) *Pursuit* – stabilizing the retina with regard to a moving object of interest. Usually, however, the Human Visual System (HVS) does not exhibit pursuits when dynamically moving targets are not a part of the interface [13][14].

In addition, some eye trackers supply information about ToT as well as user manual activity including mouse and keyboard clicks.

In this research, we concentrate on the correlation between physical effort and usability. The following metrics, which have been identified as the most important effort-based metrics, are used as a measure of the *physical* effort [1][5][6][9]-[14]:

1) *Average fixation duration*,
2) *Average saccade amplitude*,
3) *Number of fixations*,
4) *Number of saccades*, and
5) *Average eye path traversed*.

Additional commonly used metrics such as the number of keyboard clicks and the number of mouse clicks are used for identifying segments of interaction rather than classifying segments.

The effort-based software usability evaluation is divided into three phases: *Measurement, Analysis, and Assessment* [1]. In the *Measurement phase*, a group of users executes a set of tasks referred to as identical independent tasks, due to the fact that they share characteristics with an identical independent distribution (iid) used in probability theory. The tasks emerge from a single scenario; however, several parameters change from task to task in a pseudo random fashion. Hence, these tasks differ in key parameters, which prevent the users from memorizing a sequence of interaction activities. Throughout the interaction process, certain user actions such as eye movement, *ToT*, keyboard activities, and mouse activities are logged.

The *Analysis* phase involves accumulating data for several physical effort-based metrics such as the *number of saccades*, *average saccade amplitude*, *number of fixations*, *average fixation duration*, and *average eye path traversed*. Another metric is the *ToT*. The average task completion time and/or an effort-based metric are compared to a learning curve, which reflects users' mastery of software.

The final step is the *Assessment*. Using the above steps, the learnability of software systems is assessed and the point of users' mastery of the software is identified. In addition, as detailed in section D, the learnability curve is used to obtain operability and understandability of various software systems or different groups of users using the same system. Effort-based metrics provide interface designers with means to evaluate their designs [1][5].

D. The Learnability based Usability Model

Typically, as users become familiar with an application, the effort and/or the time to complete tasks, which emerge from the same scenario, become smaller or shorter [18]. Often, a graph of the averages of *Effort-On-Task* (*EoT*) or *Time-On-Task* (*ToT*), also known as effort average (E_{avg}), for the users fits well into an exponential decay curve that represents the average effort on task expended by the group of users. Figure 1 depicts a typical graph. The E_{exp} line is the effort that the interface designer expects an expert to expend in order to complete a specific task. The point where the user's effort reaches the acceptable level is the learning point L_p. The learning time (L_T) is calculated by adding the average task duration to the left of the learning point. Data to the right of the learning point relates to the amount of effort required by a trained user to complete tasks.

Understandability can be inferred from the graph by investigating the difference between the expert curve and the average user curve; while operability can be inferred from the distance between the expert curve (E_{exp}) and the *X*

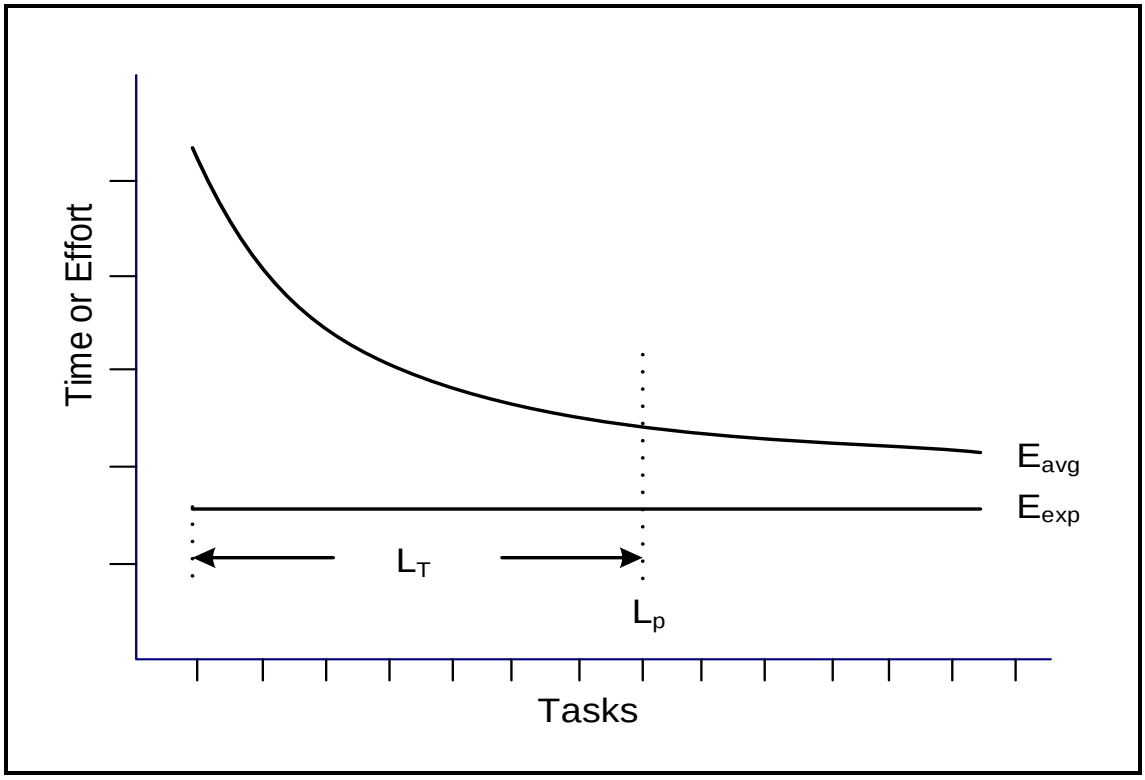


Figure 1. Learnability-based usability model

axis. Thus, the *effort-based usability model* enables the evaluation of the understandability learnability, and operability of a specific system. In addition, comparing the plots representing the results of tests with different systems or different user populations (e.g., students vs. novice employees) can be used to evaluate the relative usability of these systems [1]. This is referred to as "system A vs. system B" or "population A vs. population B" experiments [1]. Moreover, this model is further used to identify outlier tasks, which are studied to find usability shortfalls [1]. Outlier tasks are good candidates for a specific type of pinpoint analysis referred to as inter-pinpoint analysis.

E. Pinpoint Analysis

Software usability testing is one of the most expensive, tedious, and least rewarding tests to implement [1]-[5]. This perception is likely to change if the usability testing is made less expensive and more rewarding. This requires accurate means through which an engineer can identify and pinpoint issues in the software or the interface. This process is called pinpoint analysis. Pinpoint analysis is one of two types; inter-pinpoint analysis deals with identifying issues with tasks performed by the users in a specific system, whereas intra-pinpoint analysis refers to identifying issues within tasks in a specific system. For example, outlier tasks might be identified through inter-pinpoint analysis and used for intra-pinpoint analysis. This analysis can help graphical user interface (GUI) designers to make decisions about element placement on displays and determine the level of effort that is related to different widgets [1][5].

1) Inter-pinpoint Analysis

Inter-pinpoint analysis involves detecting tasks that present anomalies and identifying the reasons for these anomalies at a high level. The mouse is used as an example to illustrate inter-pinpoint analysis. In a particular task, the right mouse button helps users complete a task effectively; however, some of the users are unaware of it. It is possible that anomalies like this can be identified in inter-pinpoint analysis [1][5].

Inter-pinpoint analysis helps in identifying alternative methods to perform a task effectively with less effort; however, it does not provide users with a hint of the alternative method. Other issues like the necessity of help facilities in software are identified by the high level analysis of tasks that present anomalies.

Figure 2 is a plot of the average ToT of five subjects for seven identical but independent tasks. The X axis shows a data point for each of the seven tasks, while the Y axis shows information related to the ToT. The curve fitted to the individual bars representing the average ToT is an exponential curve that actually corresponds to the learnability model. The high correlation value ($r^2 = 0.96$) shows that the exponential curve well fits the data. Again, placing the plot of more than one systems' test in the same graph can be used for a "system/population A vs. system/population B" comparison. In this case, task-3 that does not fit well in the curve shows an anomalistic behavior and calls for further analysis and study [1][5].

2) Intra-pinpoint Analysis

Intra-pinpoint analysis is a detailed method for analyzing tasks and identifying specific issues with the software. The analysis can be done manually by watching video recordings of users' interactions with software and/or watching videos obtained from an eye-tracking device. The review helps in identifying interaction issues and areas where the user has difficulty while performing tasks. For example, the analysis might reveal that most of the users go into a state of confusion in a specific part of a task, and are searching the screen to identify the best way to proceed with

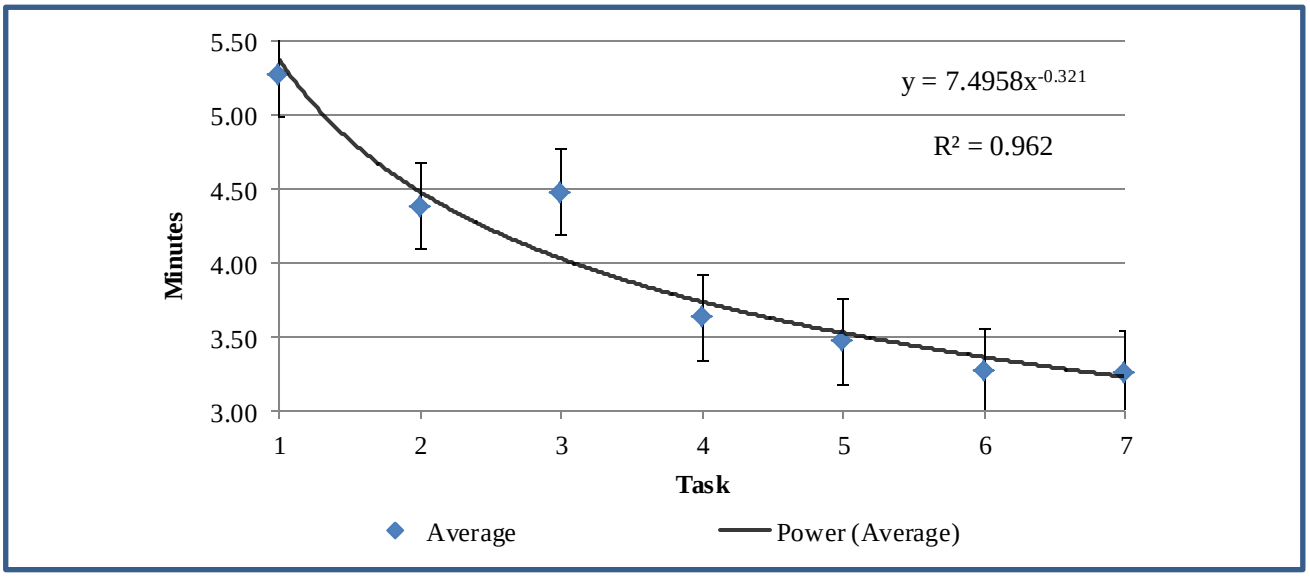


Figure 2. – Time-on-Task (TOT) for the use case of interest.

the task. This might prompt the designers to rearrange the interface where a snapshot identifies excessive effort. Clearly, the manual recording inspection is tedious and potentially expensive. An alternative is to use automatic methods utilizing pattern recognition techniques. This method eliminates the need for a person to watch the entire video in order to identify interaction issues, thereby cutting down the cost and time. It enables automatic identification of areas where the user has difficulty and marking these areas for further evaluation.

F. Pattern Recognition

One of the applications of pattern recognition is the assignment of *label*s to a given input value, or *instance*, according to a specific algorithm. An example of pattern recognition procedure is classification, which attempts to assign each input value to one of a given set of classes.

Pattern recognition is generally categorized according to the type of learning procedure used to generate the output value. *Supervised learning* assumes that training data (the training set), consisting of a set of instances that have been properly manually labeled by an expert with the correct output, has been provided. Next, a learning procedure generates a model that attempts to meet two, sometimes conflicting, objectives: Perform as well as possible on the training data, and generalize as well as possible to new data. On the other hand, *unsupervised learning* assumes the availability of training data that has not been hand-labeled and attempts to find inherent patterns that are used to determine the correct classification value for new data instances [15]-[17].

Algorithms for pattern recognition depend on several parameters, such as the type of output labels, and on the training/learning methods that are supervised or unsupervised. Additionally, the algorithms differ in the way that inference is performed. For example, inference might be based on probability, non-parametric clustering, fuzzy logic, etc. [15]-[17]. The following are various relevant pattern recognition techniques.

1) Segmentation

Pattern recognition procedures require the definition of patterns (i.e., segmentation). In this research, segments of user activities records serve as the basic patterns. A segment is defined as the time between two consecutive keyboard/mouse clicks.

2) Feature Extraction and Feature Selection

Generally, the objects that are subject to classification, i.e., the patterns (segments in the case of this research), are represented through a set of measurements (say n measurements) or characteristics referred to as features. Hence, the objects are considered as vectors in an n-dimensional space referred to as the feature space. Feature selection is a technique for selecting a subset of relevant features for building robust learning and inference models [15][16]. Feature selection algorithms attempt to reduce the dimensionality of the feature space and reduce the complexity of the recognition process by pruning out redundant, correlated, and irrelevant features. There are several feature selection algorithms, some of which are discussed below [16].

Exhaustive search is a brute-force feature selection method where all possible subsets of the features are exhaustively evaluated and the best subset is selected. The number of combinations of r objects from a set of n features is $(\frac{n!}{r!(n-1)!})$. This might result in a very large set of combinations of features to examine. Hence, generally the exhaustive search's computational cost is prohibitively high. Thus, this method is impractical if the number of features in the subset is large or the processing and evaluation time for each subset is long [11][16]. Because of the problems associated with exhaustive search, researchers resort to adopting heuristic feature selection algorithms. In this paper, there are five features of interest. This is a relatively small number of features. Nevertheless, each evaluation session requires significant computation time. Hence, due to the complexity of the evaluation process, exhaustive search is not a viable option. For these reasons, a heuristic approach is adapted.

Heuristic search refers to selecting a feature subset by making an educated guess and finding out if the selection yields good results. Otherwise, the heuristic procedure examines other subsets [16].

3) Principal Component Analysis

PCA is an unsupervised regression procedure that analyzes data samples, such as the set of training patterns, in order to identify a coordinate transformation that de-correlates the data and "orders" the information (or variance) associated with the data in the axes of the new space in a monotonically non-increasing fashion. In general, as a result of the transformation, most of the information associated with the data is concentrated in the first few components of the new space. This enables ignoring components (axes) that do not carry significant information, thereby reducing the dimensionality of the space used for pattern representation and recognition. Each principal component is a linear combination of the original variables. The principal components as a whole form an orthogonal basis for the data space [16].

The distinction between PCA and feature selection is that following the PCA the resulting features are different from the original features; they do not correspond directly to the set of measurements, and are not easily interpretable, while the features left after feature selection are simply a subset of the original features.

Following the feature selection and/or PCA, classification is applied via different methods including thresholding, discriminant analysis, decision functions, and clustering [15]-[17].

In this research, heuristic based greedy feature selection techniques as well as PCA are used to reduce the dimensionality of the data set consisting of a number of interrelated variables, such as the *number of saccades* and the *average saccade amplitude*, *number of fixations* and *average fixation duration* while retaining as much as possible of the variation present in the data set. In the case of PCA, this is achieved by transforming the data set into principal components, The principal components are then subjected to thresholding and/or clustering algorithms to find segments of excessive effort.

4) The Threshold Method

The threshold method can be used to classify input data based on a threshold value. In this research, the threshold value for each feature is the average of the values of the feature over the entire set of segments. All values greater than the threshold are placed into the "excessive effort" group while input values below the threshold are placed into the "*non-excessive effort*" group. One problem with the threshold method is that it is limited to one dimensional data. Hence, it is only applied to individual features, or a combination of features, such as linear combination or a specific component of the principle components. Clustering techniques, however, are used to efficiently classify multidimensional data.

5) Clustering

Clustering is a multi-disciplinary, widely-used, unsupervised method for classifying data. It involves the assignment of a set of patterns into subsets (called clusters) so that patterns in the same cluster are similar in some sense. To define a cluster, it is necessary to first define a measure of similarity or distance, which establishes a rule for assigning patterns to the domain of a particular cluster center. Generally, and in this paper, Euclidian distance is used as the distance measure. In Cartesian coordinates, if $p = p_1, p_2, \ldots, p_n$ and $q = q_1, q_2, \ldots, q_n$ are two points in an n dimensional space, the Euclidean distance between p and q is:

$$D(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

The Euclidean distance is used as the measure of similarity; the smaller the distance, the greater the similarity. There are several clustering algorithms, such as the hierarchical, partitional, density based, and subspace clustering algorithms [15]-[17]. In this research, however, partitional algorithms are of interest. Partitional clustering involves partitioning of n observations (patterns) into k clusters where each observation belongs to the nearest cluster. The K-means algorithm, used in this research, is a partitional algorithm that attempts to minimize the mean square distance between patterns and cluster centers, where the cluster center is the centroid of the cluster patterns. The algorithm consists of the following steps [15]:

Step 1 (seeding): Choose K initial cluster centers $z_1(1), z_2(1), \ldots, z_k(1)$. The seeds can be chosen in many different ways[13][15]. In this research, K random centers serve as seeds. The set of cluster centers at the L^{th} iteration is denoted by $z_i(L)$, where, $i = 1,2, \ldots, K$.

Step 2: At the M^{th} iterative step, distribute the patterns $\{x\}$ among the K cluster domains, using the following decision rule $x \in S_j(M)$ if $\|x - z_j(M)\| \leq \|x - z_i(M)\|$ for all $i = 1,2, \ldots, K$, $i \neq j$, where $S_j(M)$ denotes the set of patterns whose cluster center is $z_j(M)$.

Step 3: Using the results of step 2, compute the new cluster centers, such that the sum of the squared distances from all points in $S_j(M)$ to the new cluster center is minimized. The new cluster centers are given by

$$z_j(M+1) = \frac{1}{N_j} \sum_{x \in S_j(M)} x$$

For $j = 1,2, \ldots, K$, where N_j is the number of samples in $S_j(M)$.

Step 4: Repeat step 2 and step 3 until there is no change in the cluster centers, i.e., if $z_j(M+1) = z_j(M)$ for $j = 1,2, \ldots, K$, then the algorithm has converged and the procedure is terminated.

The advantage of the clustering technique is its ability to classify *excessive effort segments* by considering a number of features such as *saccade count*, *average saccade amplitude*, *fixation duration*, *and average eye path traversed*. In addition, the K-means clustering can be used to identify thresholds. MATLAB, a high-level programming language and interactive environment for numerical computation, visualization, and programming [19], has a built in function for K-means that operates exactly as described in this section. This function has been used in our project.

III. LITERATURE REVIEW

Usability is a highly researched topic with much literature available [1-6][21]-[23]. Nevertheless, extensive review did not reveal any research papers related to pinpointing usability issues (except for our prior work described in [1][9]-[12]. There are some papers on effort-based usability evaluation that are discussed below.

Tamir et al. concluded that effort and usability are related but they did not address pinpointing issues [5]. Mueller et al. use effort metrics to evaluate software usability [6]. Their method allows comparison of two or more implementations of the same application, but does not identify where exactly the problem lies. Hvannberg et al. described the design and test of a defect classification scheme that extracts information from usability problem [20], but is limited since it does not define the causes underlying usability problems. Nakamichi et al. investigate the relations between quantitative data, viewing behavior of

users, and web usability evaluation by subjects [21]. They conclude that the moving speed of the gazing points is effective in detecting low usability. Makoto et al. use a Web-Tracer to evaluate web usability [22]. Web-Tracer is an integrated environment for web usability testing that collects the operation log of users on the Web pages. However, the reasons for low usability are not identified using this approach. Our paper thoroughly addresses and resolves all of the issues listed above.

IV. EXPERIMENTAL SETUP

A. Manual Input Devices

The subject performs the tasks on a computer using a standard keyboard and a mouse as input devices. An event driven logging program is used to obtain details of mouse and keystroke activities from the operating system event queue. The program saves each event along with a time stamp into a file. The logged events are: mickeys (mouse pixels), keystrokes, mouse button clicks, mouse wheel rolling, and mouse wheel clicks.

The eye tracker used for the experiments is Tobii X120 Eye Tracker with Tobii Studio version 2.2.5 [23] as well as "in-house" developed software for estimating user's gaze. The Tobii device is a standalone eye tracking unit designed for eye tracking studies. It provides raw eye gaze positional data and is able to log mouse and keyboard events. The data collected by the eye tracker is logged into a file, which is referred to as a *log file*. The eye tracker also records a video version of the user interaction session and is referred to as a *video file*, which is very helpful in verifying experiment results. In addition, the Tobii X120 eye tracker can log mouse/keyboard clicks. The combination of the *log file* and *video file* are referred to in this paper as the *data file*.

B. Software Environment for Analysis

A software program developed in MATLAB is used to perform data analysis of the experiments reported in this paper [19]. In addition, the program is responsible for features collection and extraction.

C. Test Procedure

Experiments conducted to evaluate the capability of pattern recognition techniques to identify software usability issues are done using the steps depicted in Figure 3. As the figure shows, the main steps are: data gathering, segmentation, data reduction, feature extraction and selection. These actions are followed by several different classification techniques. The sequence of actions depicted in the figure is further described in the next three subsections: data gathering, data reduction, and identification of *excessive effort segments*.

1) Data Gathering

A group of five users executes a set of seven identical independent tasks, which emerge from a single scenario. Throughout the interaction process, certain user activities such as eye movement, ToT, keyboard, and mouse activities are logged using the eye-tracking device. According to the *learnability-based usability model*, the point at which the user's effort reaches the acceptable level is called the learning point. Based on this model, and inspection of the learning curve of the subjects, it is assumed that the users' effort reaches the acceptable level by the time they perform task-5. Hence, in this paper, task-5 of each subject is used for conducting the pinpoint analysis experiments.

2) Data Reduction

Phase-2 includes activities such as segmentation, data reduction, and feature extraction. The data logged throughout the user interaction session is used for automatic event based segmentation where the events are consecutive keyboard/mouse clicks. Metrics such as:

(a) *segment duration* (for event based segmentation),
(b) the *average fixation duration*,
(c) the *average saccade amplitude*,
(d) the *number of fixations*,
(e) the *number of saccades*, and
(h) the *average eye path traversed*

are inferred for each segment. These metrics are used to generate a feature set, which is obtained by applying data reduction programs to the data file. The features data is calculated for all features within each segment and this data is used to identify *excessive effort segments*.

2) Identification of Excessive Effort Segments

Pattern recognition techniques are applied to the feature set obtained from the data reduction process to identify segments that exhibit excessive effort. The techniques used and applied on the feature set are thresholding, K-means clustering, and PCA.

Thresholding - a threshold value is calculated for each feature in the feature set. For a given feature, all the segments that have a feature value that is less than the threshold value are classified as *non-excessive effort* segments and segments with a feature values above the threshold are considered as *excessive effort segments*. In the current research, the threshold is the average value of the feature in the segment.

K-means clustering - the segments are grouped into clusters. Based on the value of cluster centers, the cluster is classified as excessive effort cluster or non-excessive effort cluster.

PCA - the first, the second, and the third principal components of the feature data are obtained. The threshold classification, where the threshold is the average value of the first principle component computed over the set of features extracted for the current segment, is applied on the *first principal component* and K-means clustering is applied on the first, second, and third components to classify the segments into *excessive effort* or *non-excessive effort segments*.

By the end of phase-3, the software program identifies the *excessive effort segments*. To verify the results, the video

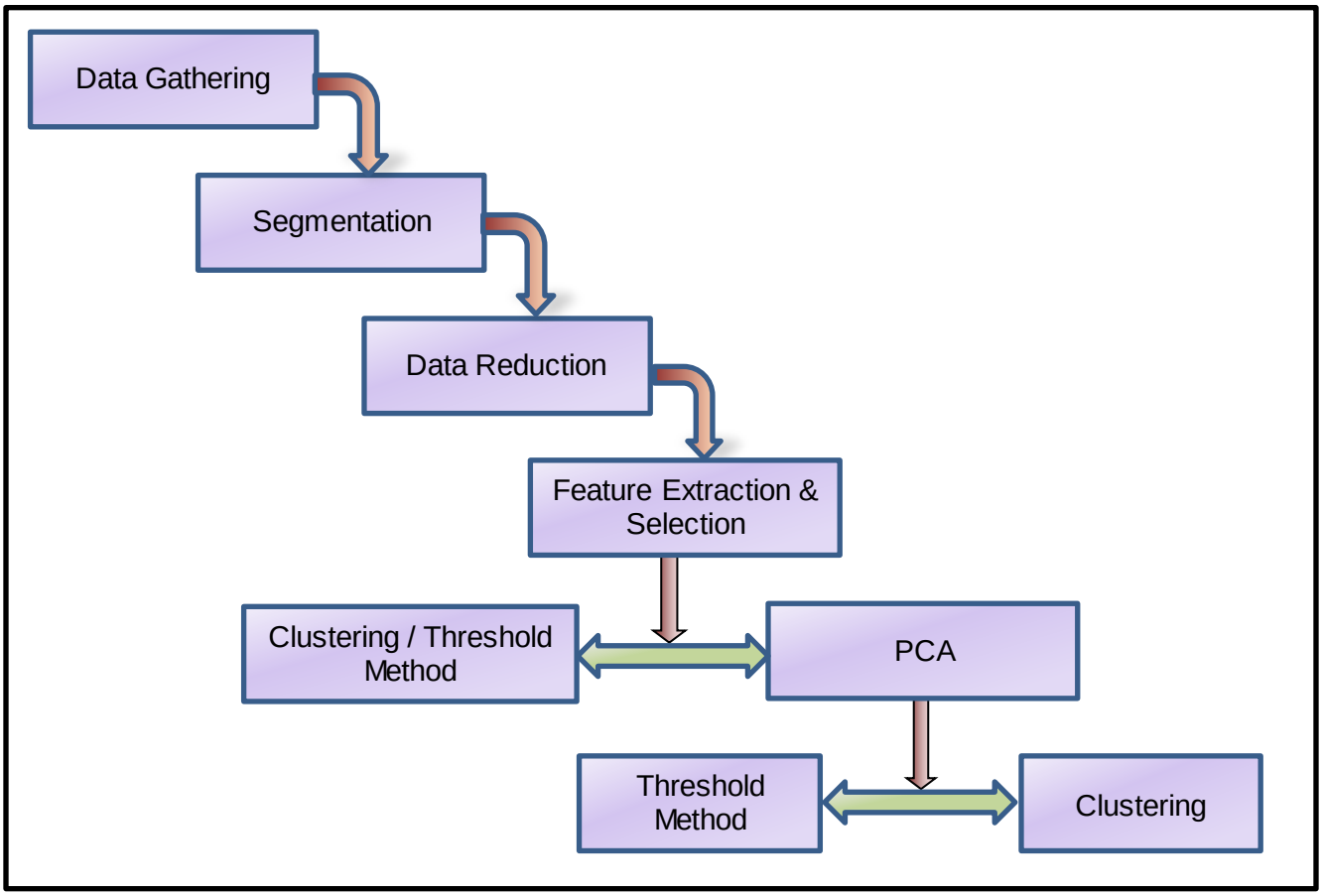


Figure 3. Experiment procedure

file is carefully watched segment by segment and classified into *excessive* or *non-excessive effort segments* manually. The manual analysis results are used as *ground truth* and serve as the input for error analysis that further supports the reliability of our results. The manual classification process of the video file is described in the following section.

D. Manual Classification

The manual classification process involves automatic event based segmentation on the entire video file. Each segment is carefully watched and classified into the following categories:

Idle behavior segments: Idle behavior is due to system response. Waiting for a progress bar to complete or waiting for a page to load are examples of idle behavior. Segments with such behaviors are classified as idle behavior segments.

Excessive effort segments: Segments without any useful user actions are classified as *excessive effort segments*. A subject looking at different components on an interface instead of the actual target component, which help in accomplishing the task, is an example of excessive effort behavior. Such behavior can be eliminated without sacrificing task completion quality.

Non-Excessive effort segments: Segments with useful action that result in task completion are classified as non-excessive segments.

Off screen behavior segments: Intervals of time where the subject's view is not within the screen for more than one second, with no meaningful user action, are classified as off screen behavior segments.

Attention segments: Segments with frequent on screen behavior, e.g., segments with very frequent mouse/keyboard clicks are classified as attention segments.

Once the video file is classified into one of the above five segment categories, the manual classification results are compared with the automatic classification results. Nevertheless, idle behavior, off screen behavior, and attention segments can be accurately identified by the software tool and are discarded from further analysis in this work. In this sense, our results are conservative as we so not measure the additional time saving obtained via the identification these types of segments.

E. Result Verification

The number of *Excessive* (*E*) *vs. Excessive*, *Excessive vs. Non-Excessive (NE)*, *NE vs. E* and *NE vs. NE* segments,

as well as related error rates, are calculated for each result file and graphs are plotted to visualize the results and enable comparing the performance of different methods and features. Classifying *NE* segments as *E* segments is regarded as false positive or type-I error.

It is assumed that all the segments classified as *Excessive Effort Segments* are due for an additional process of manual evaluation. Hence, in the case of type-I error, the software program is highlighting extra segments for further review but is not missing segments that need attention.

On a similar note, segments that show excessive effort per manual classification but are identified as *non-excessive effort segments* by the software program are regarded as false negative or type-II error segments. These segments require extra attention as the software program has misidentified segments that require the stage of manual inspection. The total time of segments classified as excessive by the software program is referred as the inspection time. It is the sum of the time interval of each of the *excessive effort segments* automatically identified by the program. In this research, type-II errors and inspection time are considered as the most important factors for analyzing experiment results.

V. EXPERIMENTS

The automatic part of the process is used to analyze the five data files by applying the different pattern recognition techniques discussed. Each of the data files is a *log file* (log of effort metrics expanded) and an eye tracking video file that contains the entire data collected by the eye tracker throughout each experiment. The following is a list of the classification experiments performed: 1) Applying the threshold method, 2) Applying heuristic feature selection and K-means clustering, 3) Using PCA, and 4) Applying K-means clustering on principal components.

Each experiment procedure is discussed in detail in the following sections.

A. Experiment 1: Applying the threshold method

In this experiment, automatic event based segmentation is applied to the eye tracking video and data file generated by the eye tracker. Next, a feature set is generated for the data file. All the segments are classified into excessive or *non-excessive effort segments* by the software program, which applies the threshold method on the following features: 1) *number of fixations*, 2) *average fixation duration*, 3) *number of saccades*, 4) *average saccade amplitude*, and 5) *average eye path traversed*.

Figure 4 presents the sequence of steps for identifying *excessive effort segments* using the threshold method.

The steps described in Figure 4 are used for identifying the *excessive* and *non-excessive effort segments*. Next, the video file segments are manually classified into *excessive* or *non-excessive effort segments* based on the specifications described above. The *excessive effort segments* identified through the software program and manual process are verified using the five data files and their corresponding video files. This step of manual classification and result verification is done in each of the experiments described in this section.

B. Experiment-2: Applying heuristic feature selection and K-means clustering

The evaluation process of a subset requires a long execution time. Hence, evaluating all the possible subsets of the feature set is prohibitively time consuming, we have adopted a heuristic greedy-based feature selection method. The following subsets have been selected: 1) *Number of fixations*, 2) *Number of saccades*, 3) *Average eye path traversed*, 4) *Number of fixations, number of saccades*, and *eye path traversed*, and 5) *Number of fixations, number of saccades, eye path traversed, average fixation duration* and *average saccade amplitude*.

Figure 5 illustrates the sequence of steps followed in identifying *excessive effort segments* using exhaustive feature selection and K-means clustering.

C. Experiment-3: Using PCA.

In this experiment, the feature set is transformed into principal components by a MATLAB function. Only the *first principal component* is considered, as it carries the most significant information related to the feature set. The *first principal component* is subjected to the threshold method for identifying segments exhibiting *excessive effort* and *non-excessive effort*. Figure 6 depicts the sequence of steps applied for identifying excessive effort, the method

Figure 4. Sequence of steps for identifying excessive effort segments using the threshold method

for identifying segments exhibiting *excessive effort*, and *non-excessive effort*.

D. Experiment-4: Applying K-means clustering on principal components

In this experiment, K-means clustering is applied to different combinations of principal components for identifying segments exhibiting *excessive effort* and *non-excessive effort*. The following constitute the feature set for this experiment: 1) 1st principal component, 2) 1st and 2nd principal components, and 3) 1st, 2nd, and 3rd principal components.

Figure 7 includes a diagram of the sequence of steps followed for identifying *excessive effort segments* using the K-means clustering on principal components.

VI. RESULTS.

In this section, the results obtained from the experiments are discussed. The results of each data file in the experiments are given in [11]. A sample of these results is presented here. For clarity, the notation used for the feature values in the graphs is presented below:

1) # Fix – denotes the *number of fixations*,
2) Avg. Fix Dur. – denotes the *average fixation duration*,
3) # Sacc – denotes the *number of saccades*,
4) Sacc Amp. – denotes the *average saccade amplitude*,
5) Eye Path - denotes the *average eye path traversed*,
6) FPC - denotes the first principal component:

A. Identifying excessive effort segments using the threshold method

In this section, we show the results obtained with data file-1. Results with other files are available in [11]. Section VII shows and analyzes the average results for all the files. The video file corresponding to data file-1 is 6.09 minutes in length. Figure 8 shows the results of an experiment using the threshold method on data file-1.

When the graph in Figure 8 is extrapolated and as seen from the *E vs. NE* bars, the feature value, *number of*

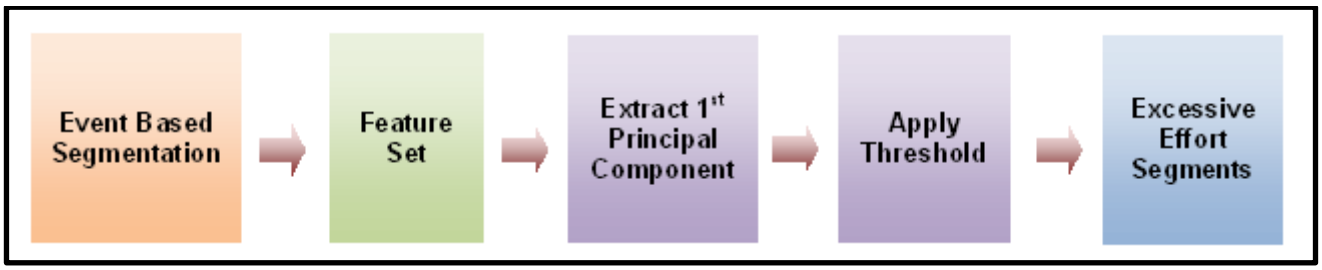


Figure 5. Sequence of steps for identifying excessive effort segments using the K-means clustering.

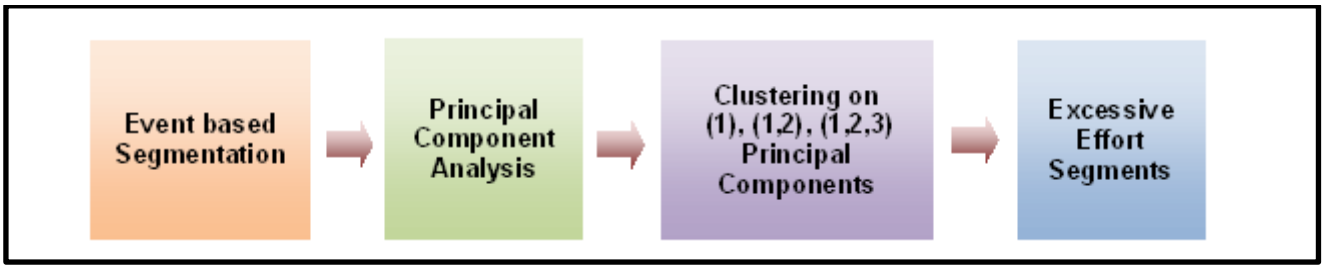


Figure 6. Sequence of steps for identifying excessive effort segments using the threshold method on the first principal component.

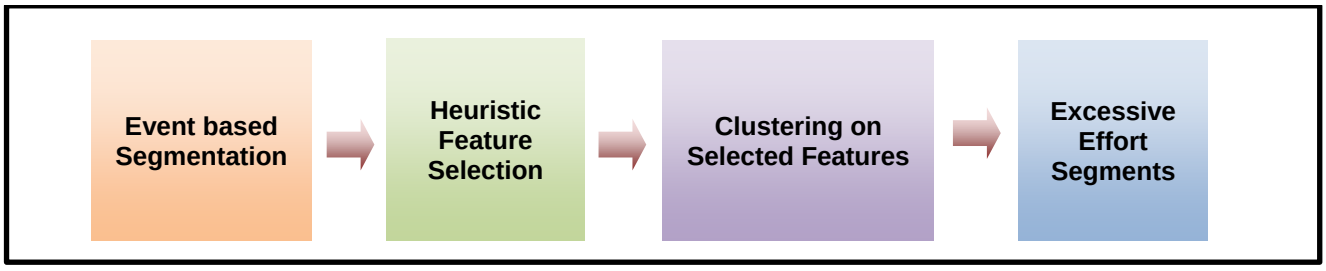


Figure 7. Sequence of steps for identifying excessive effort through K-means clustering on the 1st principal component.

fixations, demonstrates a small percentage of *E vs. NE* segments. This shows that the *number of fixations* has the least number of type-II errors. *Number of saccades* and *average eye path traversed* follows the *number of fixations* in terms of type-II errors.

Figure 9 shows the total time of segments classified as excessive by the software program and the manual process after the threshold method is applied on each of the following features: 1) *number of fixations*, 2) *average fixation duration*, 3) *number of saccades*, 4) *average saccade amplitude*, and 5) *average eye path traversed*.

The light black bars in Figure 9 represent the total video time recorded by the eye tracker. Manual classification of the video file, depicted by the dark black bars, shows 1.71 minutes of excessive effort. The *average fixation duration* and *average saccade amplitude* show a relatively low value for time of segments classified as excessive by the software program when compared with the total video time. This is depicted by the bright bars present in the figure. From Figure 9, it is observed that the percentage of type-II errors is 15.05% for *average fixation duration* and 12.9% for *average saccade amplitude*. However, the feature value with a reasonable type-II errors and lower percentage of time of segments classified as excessive is *average saccade amplitude*.

It should be noted that we are considering a 15% error of type-II as acceptable. This is explained in section VII.

B. *Identifying excessive effort segments using heuristic feature selection and K-means clustering*

In this section, we show the results obtained with data file-2. Results with other files are available in [11]. The

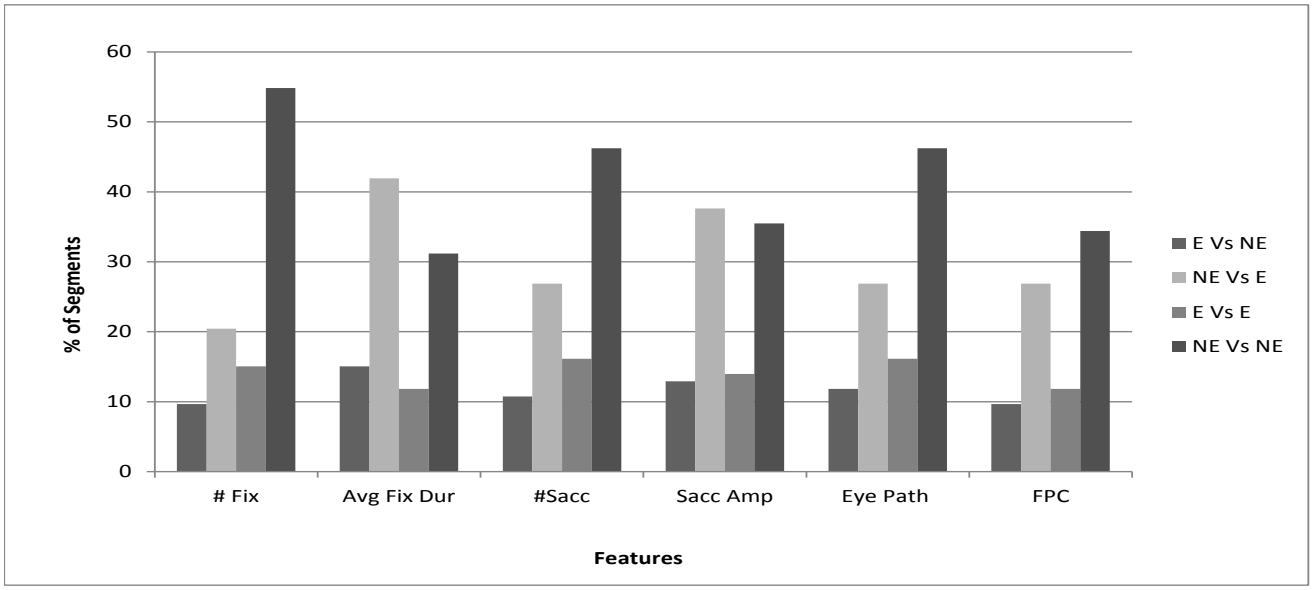


Figure 8. Percent of segments of each type (file-1, experiment-1).

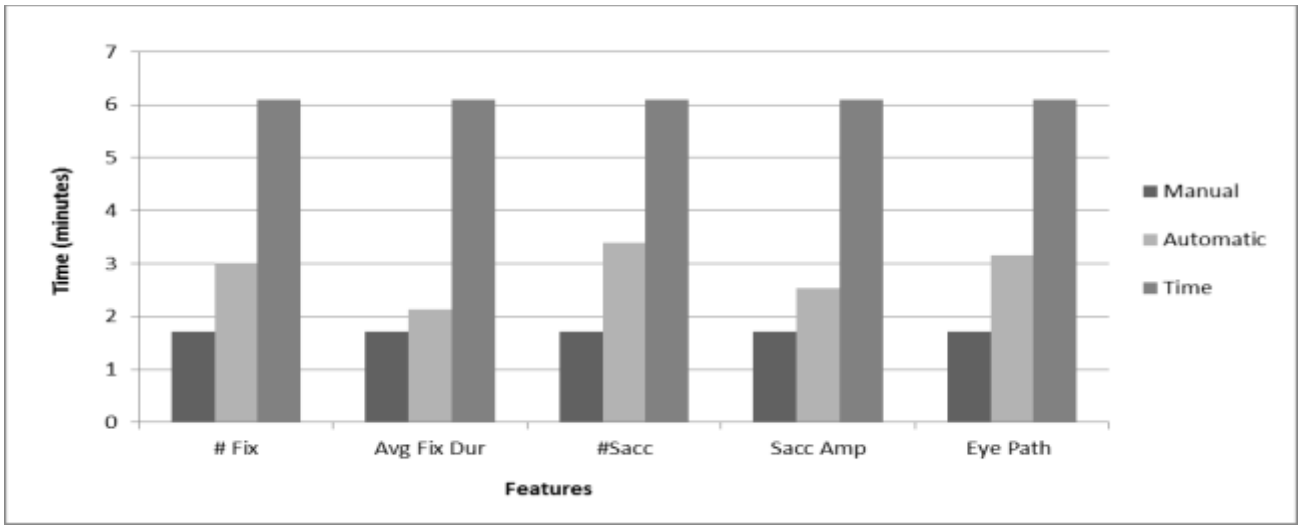


Figure 9. Total time of excessive effort segments (file-1, experiment-1).

video file corresponding to data file-2 is 3.27 minutes in length. Figure 10 shows the results of an experiment using the K-means clustering on data file-2.

When the graph in Figure 10 is extrapolated and as seen from the *E vs. NE* bars, feature subset-1 (defined above) demonstrates a small percentage of *E vs. NE* segments. This shows that the subset-1 has the least number of type-II errors. Subset-2 follows subset-1 in terms of type-II errors.

Figure 11 shows the total time of segments classified as excessive by the software program and the manual process for the above defined five feature subsets.

The light dark bars in Figure 11 represent the total time of video recorded by the eye tracker. Manual classification of the video file, depicted by the brightest bars, shows 0.36 minutes of excessive effort. Subset-3 shows a relatively low value for time of segments classified as excessive by the software program when compared with the total video time. This is depicted by the dark bars in the graph. From Figure 11 it is observed that the percentage of type-II errors is 8.5% for subset-3. Therefore, the feature value with an acceptable error of type-II and lower percentage of time of segments classified as excessive is subset-3.

C. Identifying excessive effort segments using PCA

The results of all the data files are consolidated into a single graph. Figure 12 shows the percentage of segments of each type when applying the threshold method on the *first principal component* for all five data files.

From the graph in Figure 12, it is clear that using the threshold method on the *first principal component*s produces a small percentage of *E vs. NE* segments. This means a lower type-II of errors as seen from the respective bars in the graph.

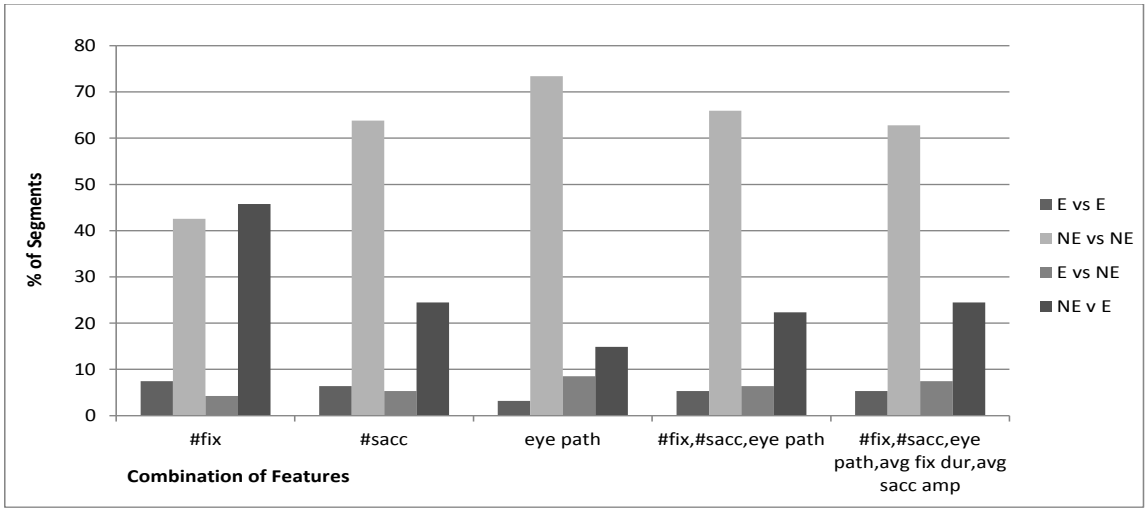


Figure 10. Percent of segments of each type (file 2, experiment-2).

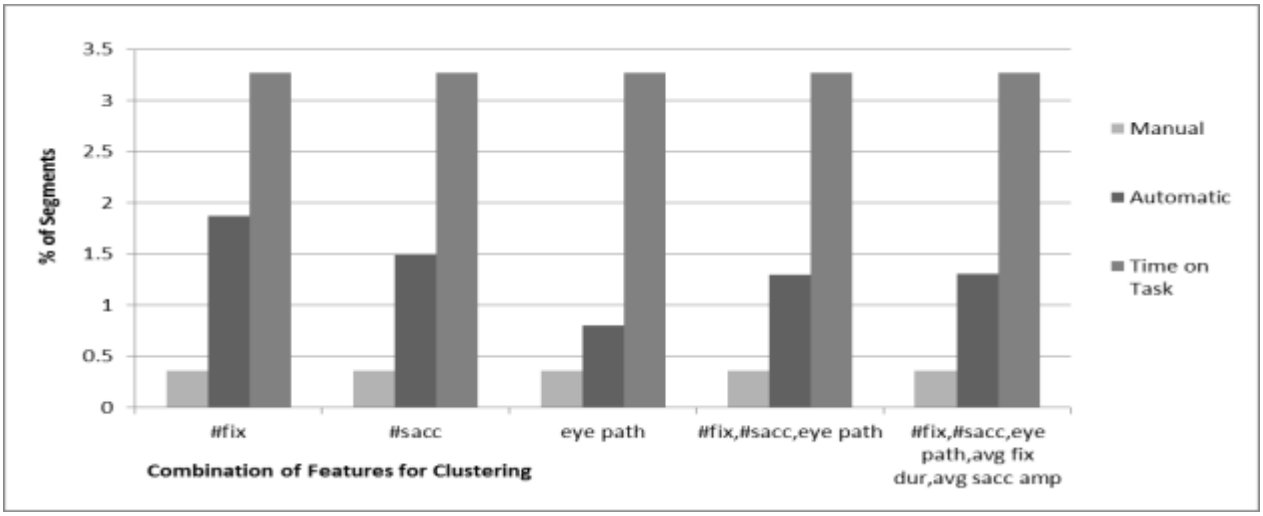


Figure 11. Total time of excessive effort segments (file 2, experiment-2).

Figure 13 shows the total time of segments classified as excessive by the software program and the manual process for the *first principal component*. The results of all five data files are plotted in a single graph.

The "light dark" bars in Figure 13 represent the total video time recorded by the eye tracker for each of the five data files. Manual classification of the video files is depicted by the dark bars. The bright bars represent the total time of video classified as excessive by the software program. The percentage of time of segments classified as excessive is relatively high when applying thresholding on the first principal component.

D. *Identifying excessive effort segments using K-means clustering on principal components*

In this section, we show the results obtained with data file 3. Results with other files are available in [11]. The video file corresponding to data file 3 is 3.8 minutes in length. Figure 14 shows the percentage of *E vs. E, E vs. NE, NE vs. NE, and NE vs. E* segments for the features mentioned in the experiment description.

From the graph in Figure 14, it is clear that all three features have same percentage of *E vs. E*, *E vs. NE*, *NE vs. NE* and *NE vs. E* segments. This signifies that most of the information is concentrated in the *first principal component*

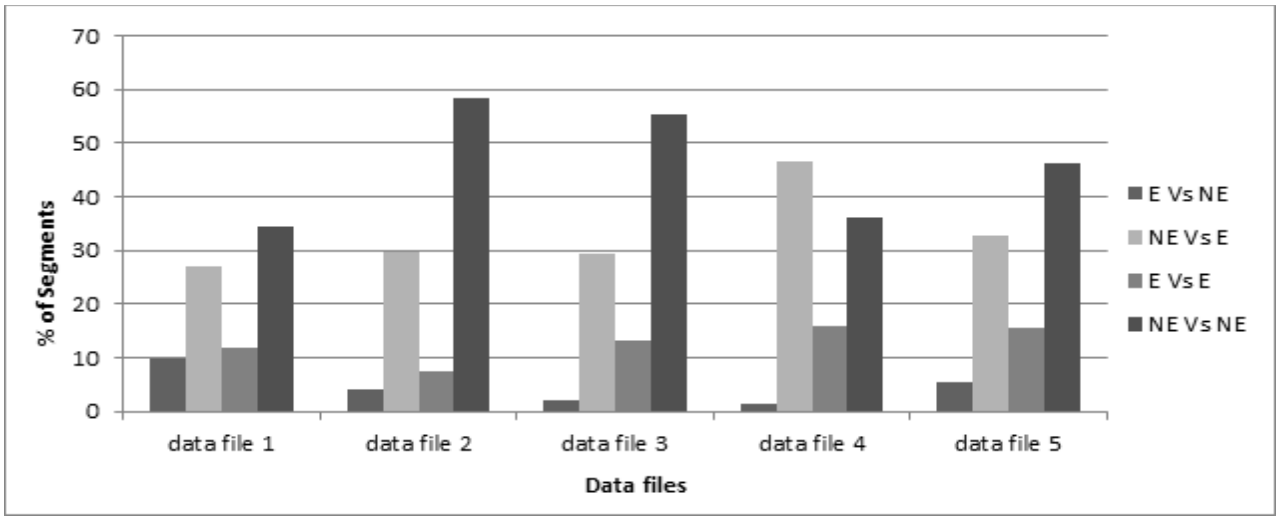


Figure 12. Percent of segments of each type (experiment-3).

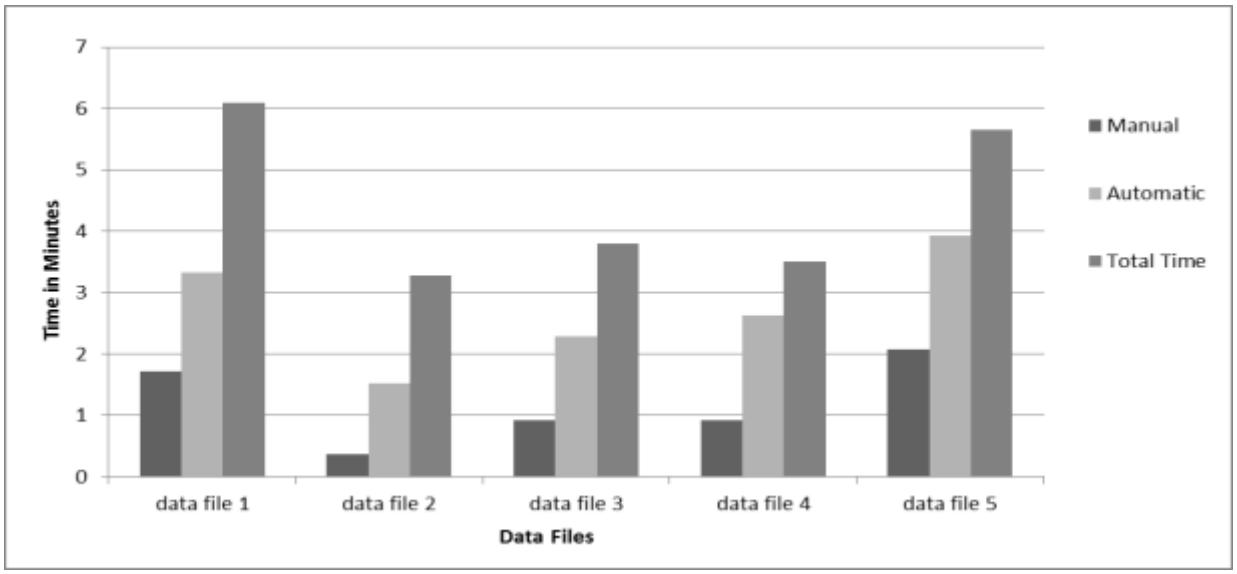


Figure 13. Total time of segments classified as excessive (Experiment-3).

and all the feature values have the same percentage of type-I and type-II errors.

Figure 15 shows the total time of segments classified as excessive by the software program and the manual process for the three features.

The "light dark" bars in Figure 15 represent the total video time recorded by the eye tracker. Manual classification of the video file, depicted by the dark bars, shows 0.92 minutes of excessive effort. The automatic classification of the video file for all three features shows 1.95 minutes of excessive effort time, which is depicted by the light bars in Figure 15. All the features have an acceptable value of type-II errors at 8.57%.

The entire set of experiments including all the data files is detailed in [11].

VII. RESULT EVALUATION

In this section, we evaluate and discuss the results of the experiments conducted in this work. Our criteria for success are based on 1) the number of type-II errors and 2) the minimal time to investigate the usability issues with an acceptable level of type-II errors. Based on discussions with several engineers in the company sponsoring this work and other companies, we are assuming that 15% of error of type-II is the upper bound for being considered as acceptable. This is also consistent with a two-step approach where after a first pinpoint analysis stage, which allows for high rate of errors but provides significant reduction in evaluation time, the errors identified are fixed, leading to a more rigorous pinpoint analysis with lower error bound. The results are evaluated based on the performance of each pattern recognition method on individual features. In addition, the

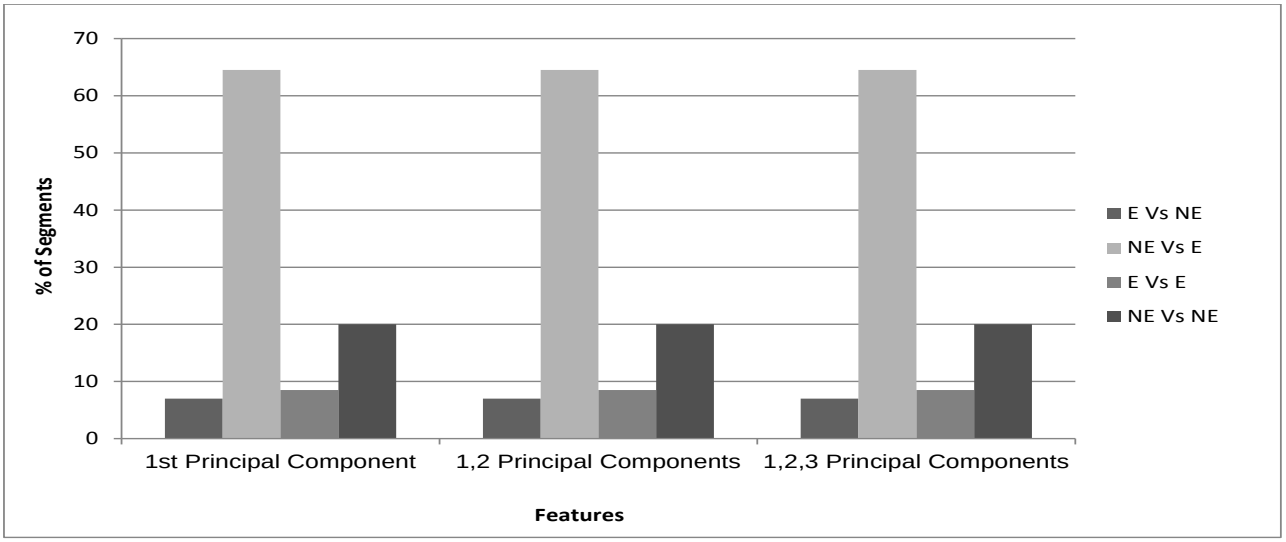


Figure 14. Graph of percentage of segments of each type (file 3, experiment-4).

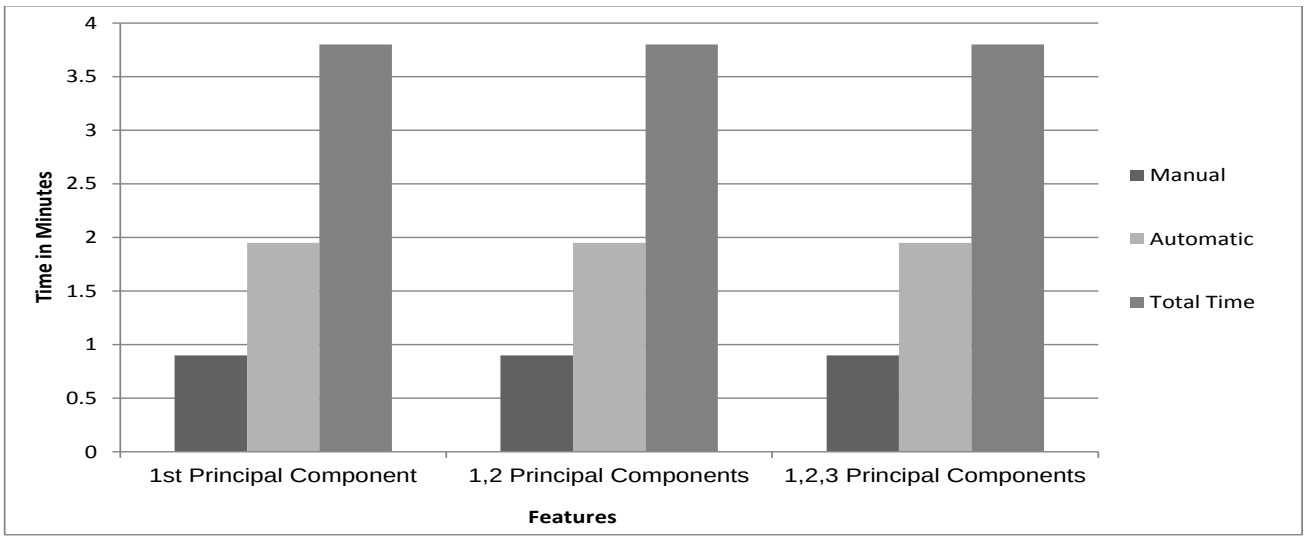


Figure 15. Total time of excessive effort segments (file 3, experiment-4).

overall performance of each pattern recognition method is evaluated.

Tables I to IV summarize the results of the experiments. An additional set of tables, which contains the entire results, can be found in [11].

A. Applying the threshold method

The following observations are derived from Table I:

1. The results of Table I show that the threshold method on the feature value, *number of fixations*, gives good results in terms of type-II errors but, the average inspection time is relatively high when compared to other feature values. The average value of type-II errors for the *number of fixations* is 3.3%. The *average saccade amplitude* and the *average eye path traversed* follow the *number of fixations* in terms of type-II errors.
2. A threshold on the *average fixation duration* performs well in terms of minimal inspection time with an acceptable value of 9.8% for type-II errors.
3. A feature value with minimum number of total errors is *average eye path traversed*. This feature value is a good choice when inspection time is not a crucial factor.
4. The inspection time is not completely correlated to type-I errors. In the case of *average fixation duration*, the inspection time is 1.67 minutes with 29.4% of type-I errors. On the other hand, the *average saccade amplitude* with almost the same percentage of type-I errors has higher inspection time than *average fixation duration*.
5. The values of the average number of *excessive effort segments* for all features are in close proximity to each other. However, the percentage of type-I and type-II errors differs invariably. Indicating that the segments classified as excessive are different for each feature value.
6. Despite the fact that the percentages of total errors for each feature value are in close proximity to each other, the inspection time varies. This delineates that the segments classified as excessive are different for each feature value.

B. Applying heuristic feature selection and K-means clustering.

The following observations are derived from Table II:

1. The results of Table II show that the K-means clustering on the feature subset - *number of fixations, number of saccades, average eye path traversed, average fixation duration, and average saccade amplitude*, gives good results in terms of type-II errors with an average value of 5.4%.

TABLE I. AVERAGE VALUES OF EXPERIMENT-1 RESULTS.

Feature value	avg. # of excessive effort segments	avg. total no. of segments	avg. % type- I errors	avg. % type- II errors	avg. % of total errors	avg. Inspection time (minutes)	avg. Inspection time as a % of total time
# Fix	17.2	95	28.4	3.3	31.7	2.7	62.1
Avg. Fix Dur.	18.2	95	29.5	9.9	39.4	1.6	37.4
#Sacc	32	95	21.8	10.5	32.2	2.9	64.1
Sacc Amp.	17.6	95	29.1	4.6	33.7	2.5	56.4
Eye Path	17.8	95	25.7	5.1	30.8	2.6	57.7

TABLE II . AVERAGE VALUES OF EXPERIMENT-2 RESULTS

Feature value	avg. # of excessive effort segments	avg. total no. of segments	avg. % type -I errors	avg. % type -II errors	avg. % of total errors	avg. Inspection time (minutes)	avg. Inspection time as a % of total time
#fix	29.1	95	27.2	6.6	33.9	2.4	56.2
#sacc	23.5	95	17.8	8.9	26.7	2.0	45.1
eye path	19.7	95	18.0	10.1	28.1	1.6	37.5
#fix, #sacc, eye path	23.2	95	18.3	8.6	26.9	1.9	44.5
#fix, #sacc, eye path, avg. fix dur., avg. sacc amp.	29.2	95	32.6	5.4	38.0	2.5	56.3

However, the average inspection time is relatively high when compared to other feature values. The *number of fixations* follows the above identified feature value in terms of type-II errors.

2. Clustering using the *average eye path traversed* performs well in terms of minimal inspection time with an acceptable value of 10.1% for type-II errors.
3. A feature value with minimum number of total errors is the *number of fixations*. This feature value is a good choice when the inspection time is not a crucial factor.
4. The average number of *excessive effort segments* for the *number of fixations* and the feature subset with the following features: *number of saccades*, *average eye path traversed*, *average fixation duration*, *average saccade amplitude* are the same. However, the inspection times vary. Indicating that the segments classified as excessive are different for each feature value.
5. Unlike the results of the threshold method, the percentages of total errors for each feature value vary by a wide margin when applying the K-means clustering on different feature subsets.

C. Using PCA

The results summarized in Table III are compared with the results obtained from Experiment-1 to compare the performance of the threshold method on the *first principal component* with the performance of thresholding on all the other features including the *number of fixations*, the *average fixation duration*, etc. Experiment-1 result evaluation shows that the feature value, *number of fixations*, gives good results in terms of type-II errors. The average percentage of type-II errors for the *number of fixations* is 3.3%, whereas it is 4.1% for the *first principal component*. Initially, the *average saccade amplitude* and the *average eye path traversed* succeeded the *number of fixations* in terms of performance. However, the new results place the threshold on the *first principal component* right after the *number of fixations* with respect to type-II errors.

The inspection times for the first principal component and for the *average fixation duration* are 2.7 and 1.6 minutes, respectively. A threshold on the *average fixation duration* performs better than the *first principal component* in terms of lower inspection time and an acceptable 9.8% for type-II errors.

D. Applying K-means clustering on principal components.

Table IV shows the average values of all the features used in Experiment-4 over the five data files. The average type-II error is very high when using the K-means on the principal components. The average inspection time is only 1.96%. When taking type-II errors also into consideration, this method is not suitable to identify *excessive effort segments*.

Of all the pattern recognition methods used, a threshold on *number of fixations* yields the best results in terms of type-II errors with a reduction of more than 40% in manual inspection time and is followed by a threshold on the *first principal component*. The K-means clustering on the feature subset with the features: 1) *number of fixations*, 2) *number of saccades*, 3) *average saccade amplitude*, 4) *average fixation duration*, and 5) *average eye path traversed* ranks third.

The K-means clustering on the *number of saccades* yields the best results and precedes the threshold method on *average fixation duration* in performance.

.

TABLE III. AVERAGE VALUES OF EXPERIMENT-3 RESULTS

Feature value	avg. # of excessive effort segments	avg. total no. of segments	avg. % type -I errors	avg. % type- II errors	avg. % of total errors	avg. Inspection time (minutes)	avg. Inspection time as a % of total time
1st principal components	16.6	95	27.5	4.1	31.6	2.7	61.2

TABLE IV. AVERAGE VALUES OF EXPERIMENT-4 RESULTS

Feature value	avg. # of excessive effort segments	avg. total no. of segments	avg. % type- I errors	avg. % type- II errors	avg. % of total errors	avg. Inspection time (minutes)	avg. Inspection time as a % of total time
1st, 2nd & 3rd principal components	28.6	95	24.4	12.6	37.0	2.0	43.6

.

VIII. CONCLUSIONS AND FUTURE RESEARCH

The framework presented in this research enables software developers to efficiently identify usability issues and deficiencies in numerous types of applications thereby optimizing the time spent on software-usability testing and validation.

Excessive effort segments, which typically relate to usability issues, are identified by applying pattern recognition techniques, such as K-means clustering algorithm, thresholding, PCA, and feature selection. The analysis of the experiments conducted in this paper shows that the time taken for software usability testing can be reduced by 40% or more.

In this research, the time between two consecutive keyboard/mouse clicks by a user is considered as a segment and serves as the basic pattern for the pattern recognition techniques. Equal time slicing of user's software interaction session can be used instead and the performance results can be analyzed and compared with the results from this research.

Further refinement of pattern recognition techniques can be pursued to minimize errors and inspection time. Also, more focus can be placed on the criteria for manual classification of video segments thus allowing *excessive effort segments* to be identified more accurately in the first place.

Another direction for future research is to automate some of the manual steps in this process. This can include software that automatically logs the data from users' interaction session, manipulates the data, and without human intervention, identifies the *excessive effort segments*. This can significantly reduce time taken for the usability testing.

In this work, we have concentrated on pattern recognition techniques that do not rely on human intelligence. Hence, the results are generated using non-supervised learning procedures. A surrogate approach can use supervised learning procedures. This involves conducting experiments using training data sets to manually arrive at an archetype that can be applied on any data set to generate the output.

Finally, we plan to investigate the utility of dynamic UI, which adapts to the user experience. For example, widget placement might change based on usage patterns. Pinpoint analysis is expected to be a crucial tool for evaluating the effectiveness of the dynamic interface approach for identifying related deficiencies.

ACKNOWLEDGMENT

This research was funded in part by Emerson Process Management [24], an Emerson business.

REFERENCES

[1] D. K. V. Dasari, D. E. Tamir, O. V. Komogortsev, G. R. LaKomski, and Carl J. Mueller, "Pinpoint analysis of software usability," ICCGI 2013, The Eighth International Multi-Conference on Computing in the Global Information Technology, Nice, France - July, 2013, pp. 66-71.

[2] J. S. Dumas and J. C. Redish, "A Practical Guide to Usability Testing," OR, USA, Intellect Books., 1999.

[3] J. Nielsen, Usability Engineering, San Francisco, Boston, Academic Press, 1993.

[4] J. Rubin and D. Chisnell, Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests, Indianapolis, Wiley Publishing, Inc., 2008.

[5] D. E. Tamir, C. J. Mueller, O. V. Komogortsev, "A learning-based framework for evaluating software usability," *the ARPN Journal of Systems and Software*, June 2013, pp. 65-77.

[6] C. J. Mueller, D. E. Tamir, O. C. Komogortsev, and L. Feldman, "Using designer's effort for user interface evaluation," *IEEE International Conference on Systems, Man, and Cybernetics*, Texas, USA, October 11, 2009, pp. 480-485.

[7] ISO/IEC 9126-1: 2001, Software Engineering-Product Quality, Part-1, Quality Model, Geneva, Switzerland: International Standards Organization, 2001.

[8] ISO/IEC 9126-1: 2001, Software Engineering-Product Quality, Part-2, External Metrics, Geneva, Switzerland: International Standards Organization, 2001.

[9] D. E. Tamir, O. V. Komogortsev, and C. J. Mueller. "An effort and time based measure of usability," 6th Workshop on Software Quality, 30th International Conference on Software Engineering, Leipzig, Germany, 2008, pp. 35-41

[10] D. E. Tamir, et al. "Detection of software usability deficiencies," International Conference on Human Computer Interaction, FL, 2011, pp. 528-536.

[11] D. K. V. Dasari, Pinpoint analysis of software usability, Thesis Report, Texas State University, Computer Science, December 2012.

[12] C. Holland, O. V. Komogortsev, D. Tamir. "Identifying usability issues via algorithmic detection of excessive visual search," Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI), Austin, TX, 2012, pp. 1-10.

[13] A. Poole and L. J. Ball, Eye Tracking in Human-Computer Interaction and Usability Research: Current Status and Future Prospects, Encyclopedia of Human Computer Interaction: Idea Group, 2004.

[14] M. A. Just and P. A. Carpenter, "Eye fixation and cognitive processes," Cognitive Psychology, vol. 8, 1976, pp. 441-480.

[15] J. T. Tou and R. C. Gonzalez, Pattern Recognition Principles, Reading, MA: Addison-Wesley Publishing, Inc., 1974.

[16] R. O. Duda, P. E. Hart, and D. G. Stock, Pattern Classification, 2nd Ed., Indianapolis, Willey International, 2001.

[17] D. E. Tamir and A. Kandel, "The pyramid fuzzy c-means algorithm," International Journal of Computational Intelligence in Control, 2 (2), 2012 pp. 65-77.

[18] H. Ebbinghaus, Memory: A Contribution to Experimental

Psychology, 1885, http://psychclassics.yorku.ca/Ebbinghaus/memory3.htm, retrieved June 2014.

[19] Anonymous, "MATLAB Product Help," MATLAB, 2013, http://www.mathworks.com/help/, retrieved June 2014.

[20] E. T. Hvannberg and C. L. Lai, "Classification of usability problems (CUP) scheme," Nordic conference on Human-computer Interaction, Oslo, Norway, 2006, pp. 655-662.

[21] N. Nakamichi, S. Makoto, and S. Kazuyuki, "detecting low usability web pages using quantitative data of users' behavior," Proceedings of the 28th international conference on Software engineering, New York, NY, 2006, pp. 569-576.

[22] S. Makoto, N. Noboru, H. Jian, S. Kazuyuki, and N. Nakamichi. "Webtracer: A new integrated environment for web usability testing," 10th Int'l Conference on Human - Computer Interaction. Crete, Greece, June 2003, pp. 289-290.

[23] Anonymous, "Tobii X60 & X120 Eye Trackers: User Manual," Tobii, 2013, http://www.tobii.com/Global/Analysis/Downloads/User_Manuals_and_Guides/Tobii_X60_X120_UserManual.pdf, retrieved June 2014.

[24] Anonymous, "Emerson Process Management," Emerson, 2014, http://www.emersonprocess.com, retrieved June 2014.

A Detailed Description of the EC^{2}M Project: Exploiting Ontologies for the Automatic and Manual Documents Classification in Industrial Enterprise Content Management Systems

Daniela Briola
DIBRIS, Genoa University
Via Dodecaneso 35, 16146 Genoa, Italy
daniela.briola@unige.it

Alessandro Amicone
GFT Italia S.r.l.
Via Cesarea, 2/45, 16121 Genoa, Italy
alessandro.amicone@gft.com

***Abstract*—Enterprise Content Management (ECM) systems represent a crucial aspect in the efficient and effective management of large-scale enterprises, in particular for those made up of several sites distributed all over the world. The increasing number of documents to be managed, the problems related to the sharing of private information between commercial partners, the need for semantically describing the contents of shared documents have pushed researchers to find new techniques and solutions to deal with these challenges. We already presented the high level description of a joint project of the Department of Informatics, Bioengineering, Robotics and System Engineering of the University of Genoa, Italy, and two companies, Nacon (member of Sempla Group, now part of the GFT Group) and Nis, to create an improved ECM system (named EC^{2}M) exploiting ontologies to better classify, retrieve and share documentation among different sites of the involved companies: in this paper, we give a more detailed description of the project, with respect to its modules and to the underlying ontology used to classify documents. We present the automatic documents classification algorithm too, with an example of its execution. The developed system, which was born from a real industrial need, is currently used by GFT Italy to manage and share its documents among more than 600 users distributed in many different geographical locations and, thanks to the ontology, the semantic tagging process and the automatic documents forwarding have been successfully achieved. This joint project proves how a more formal representation of the documents domain can effectively improve the standard way of classifying and retrieving documents in real industrial scenarios, representing a winning collaboration between university and industry.**

Keywords– Ontologies, Semantic Classification, Knowledge Representation, Industrial Application, Automatic Documents Classification.

I. INTRODUCTION

This paper presents in detail the "Enterprise Cloud Content Management (EC^{2}M)" system, that was previously described in [1]: it is an "Enterprise Content Management" system deployed over a cloud platform, improved with the capability of semantically tagging the documents using an ontology and exploiting context information from mobile devices. This paper is an extended version of the previous one, and presents new information about the architecture and the implementation of the system modules, about the underlying ontology and about the classification algorithm used to automatically tag documents. Figures and results have been updated too, to reflect the actual running system.

The international Association for Information and Image Management (AIIM), the worldwide association for enterprise content management, defined the term "Enterprise Content Management" in 2000, but it has been updated many times to adhere to the continuous new market needs. The more recent definition is: *Enterprise Content Management (ECM) is the strategies, methods and tools used to capture, manage, store, preserve, and deliver content and documents related to organizational processes. ECM covers the management of information within the entire scope of an enterprise whether that information is in the form of a paper document, an electronic file, a database print stream, or even an email* [2]. ECM is an "umbrella term" covering document management, web content management, information search, collaboration, records management and many other tasks, but it is primarily aimed at managing the life-cycle of information, from initial publication or creation to its disposal, to preserve a company's internal (often unstructured) information, in all of its forms.

Therefore, most ECM solutions focus on Business-to-Employee (B2E) systems, but nowadays, thanks to the improvement of the IT capabilities and because of the increasing users' need to classify documents according to their meaning, these systems have grown in complexity and often integrate modules to exploit more structured information, taxonomies, dictionaries and so on.

This trend is identified both in the industrial area ([3], [4]), where the focus is usually on improving already existing products and on increasing their usability, efficiency and functionalities, both in the academic field, where the focus is more on studying new knowledge representation formats and their exploitation in automatic data analysis, classification and storage for automatic reasoning or user centric services (see next Section).

Many vendors are offering products in this area, starting from the commercial ones (Microsoft, IBM and Oracle) moving to many powerful open source solutions (for example

Alfresco [5], Plone [6] and SenseNet [7]). The new trend in this field is to create ECM systems that can automatically extract information from documents to classify them or add a semantic layer to tag documents in a more structured and interesting way: this is the area where the EC^{2}M project is located.

The problem of classifying, retrieving and sharing documents among users and companies pertains to the research field of knowledge sharing, whereas the problem of semantically tagging documents pertains to that of the knowledge representation. Both fields are relevant both from an academic and an industrial viewpoint, and this motivates the joint academic-industry EC^{2}M project. In EC^{2}M, we used ontologies as a way to structure information describing documents and their content.

Many similar studies and projects have been conducted in this area: an example of a commercial ECM system semantically enriched is SmartLogic [8], which offers an automatic classification, based on an automatically-extracted taxonomy of documents. Many open source systems have been developed to integrate semantic services in the document/information management, among which H-DOSE [9], OPEN-CALAIS [10] and APACHE STANBOL [11]. Even if they are not ECM systems according to the standard definition, they deal with very similar problems.

We decided to exploit for our system some of these open-source systems (as described later), to be able to freely combine them to get an improved ECM system. None of the available systems offered a complete solution to our problem, so we adopted a mesh-up approach, based on open source softwares, and we then integrated in it an ad hoc ontology, shared among the different nodes of the network, to model the documents types and their content.

The system offers a publish/subscribe service and is based on a cloud platform. It also takes exploits contextual information on the location and device used by the user to implement context-awareness. In this way, the resulting system takes advantage of well known and high quality open source softwares and of a powerful cloud platform, and enriches the available solutions with new techniques not used in standard ECM systems.

EC^{2}M is thus a concrete industrial-academic example of how these technologies can be composed and improved to create a new powerful system, which can be actually used by enterprises.

The rest of the paper is organized in six sections: Section II presents the state of the art regarding similar academic systems, Section III describes the EC^{2}M system, Section IV presents the ontology, Section V shows the actual implemented system, Section VI describes the automatic classification algorithm used in the system and, finally, Section VII concludes the paper.

II. State of the art

If we look at academic research, many studies underline how a semantic and structured representation of the domain (with ontologies or similar techniques) can improve systems like CMSs, where documents and data must be stored and classified, manually or automatically, so that users can easily find what they are looking for.

For example we can cite [12], describing the Rhizomer CMS, which tags its items using semantic metadata semi-automatically extracted from multimedia sources, or [13], which proposes a framework to manage and share written information contents using an ad-hoc knowledge model for an industrial research center, or [14], which presents an open architecture framework based on the open-source CMS OpenCMS and a Java-based web management system for learning objects, which were derived from the instructional materials used in several postgraduate courses.

More recently, many other analyses and examples have been realized.

In [15], a set of tools have been developed to semi-automatically explicate the semantics of a content repository into a knowledge-base and to establish semantic bridges between this knowledge-base and the content repository (the tools set is complemented with a search engine that makes use of the explicated semantics). In [16], a semantic-based content abstraction and annotation approach is proposed: based on this approach, a semantic-driven content management environment has been developed to deliver the right content to the right user at the right time. According to the authors of [17], dealing with problems and possible architectural solutions in managing heterogeneous oceanographic data are reported, a careful employment of ECM systems may be beneficial in that setting, with no need to adopt complex ad hoc solutions that are difficult to maintain by personnel not specifically skilled in data-handling techniques. A data model to support the storage of refined data in structured repositories is developed and presented in that paper as well. In [18], the described approach is to model context (the public documents of a Public Administration) in an ontology and to use that ontology to infer content-related metadata to be associated to the documents, avoiding to do this operation manually.

Our project is mainly focused on the knowledge representation and sharing research areas, but it takes into account software design problematics too, like those found in user-centric and context aware systems development: many academic research works dealing with these topics exist, proposing different solutions and approaches.

For example, in [19] the Multiagent paradigm is used as underlying architecture to develop distributed intelligent ubiquitous systems where applications and services can communicate in a distributed way with intelligent agents, even from mobile devices, while in [20] the authors present a framework to develop context-aware dialogue systems that dynamically incorporate user specific requirements and preferences as well as characteristics about the interaction environment, in order to improve and personalize web information and services. In [21], a distributed architecture called inContexto, which uses mobile phones, is used to infer physical actions performed by users starting from user context information. Starting from the assumption that the human context within which a software system will operate is fundamental for its requirements, in [22] a framework based on the socio-psychological Activity Theory and its analysis of human contexts is presented.

We also found systems that exploit ontologies to improve knowledge sharing, but dealing with domains that are completely different from ours (for example [23], which is a semantic television content management system based on ontologies) or relying on different architectures (for example [24], where an Ontology Server (OS) component is created to be used in a distributed content management grid system): even if the domains or the proposed solutions are different, the underling problem still remains the same, proving that it is still open and studied.

III. THE EC^{2}M SYSTEM ARCHITECTURE

The Enterprise Cloud Content Management (EC^{2}M) system was born from the collaboration among the Department of Informatics, Bioengineering, Robotics and System Engineering of the University of Genoa and two outstanding IT Italian enterprises, namely Nacon (member of the Sempla Group, now GFT Italy, specialized in the design and implementation of complex systems, ECMs, and process management) and Network Integration & Solutions (Nis), specialized in the design and development of network products and services for businesses, public administration and end-users.

The developed system is a Content Management System that aims to automatically classify documents with respect to an ontology defining the possible predefined tags (and, at the same time, to help users in semantically tagging documents that they are manually inserting in the system): these documents will be then shared among different partners (called "Nodes of the EC^{2}M network", which are companies or companies' sites, which need to collaborate and agree on using the common ontology to tag documents), located in various physical locations, in an automatic way.

The types of documents and their possible contents are modeled using an ontology that formally describes them; the ontology instances are used to tag the documents with semantic information. Every user in the system is able to subscribe to a set of "interests" (chosen from the instances in the ontology) so that when a new document is inserted in the EC^{2}M network and tagged (manually or automatically) with terms from the ontology, those users that are interested in those terms are proactively informed that a new document is available.

The routing process, which in our case is the process of informing the Nodes in the EC^{2}M network about the existence of new documents and of consequently sharing them, is demanded to a Central Router Node.

A software module manages the context (location and device) where the user is acting, to give the user a subset of the information he needs considering the device he is working on.

The system is deployed over the Cloud Amazon Web Services (AWS) platform (using it as an "Infrastructure as a Service"), which is a good compromise between cost and performances. This solution allows to simply scale the number of nodes in the EC^{2}M network or to scale the physical resources used to manage the network, to get better performances, and at the same time grants a reliable Central Router implementation, avoiding the standard "single point of failure" problem related to the centralized routing architecture.

Anyway, the EC^{2}M system has been designed to run on a private physical network too.

The EC^{2}M system offers "internal services" to individual nodes (corresponding to an enterprise site) and "external services" to let nodes interact. Looking at individual nodes, the so called "semantic publish/subscribe" service allows every user to declare the arguments he is interested in, chosen from those described in the ontology, and then makes available (globally on the network and locally to the node) the documents matching the subscribed interests.

Looking at the complete network, that is, at the services connecting different nodes, the aim of the system is to allow users from different nodes to be informed of documents, on other nodes, which are interesting for them. This is where the ontology comes into play: sharing interesting documents across nodes is in fact possible because the nodes share a common ontology (or a subpart of it).

Every node in the network may have privacy policies, because not every document of a node should be read by all the other nodes: maybe only some information as title, abstracts, etc. can be shared. These policies are managed by the Nodes. Issues related to policy management are out of scope of this paper and are not described here.

Every document is characterized by a set of standard attributes (or tags), like its Name, Creation date, Abstract and so on, whereas the document type is chosen from the ontology: then the user can add other tags in a manual or semi-automatic way (see more details in Section V), selecting the values from the instances of the ontology and driven by their relationships.

The EC^{2}M system can be "instantiated" many times, to be useful for different groups of enterprises (that is, for new enterprises' networks): a new ontology, describing types of documents and their possible contents must be created, but the overall structure and behavior remain the same. In this sense, EC^{2}M is "parametric" in the used ontology.

The core portion of the functional requirements (services) specification of EC^{2}M system has been created using the method proposed in [25], and consisted of: (1) an UML Use Case Diagram, (2) the Use Case descriptions, (3) a glossary, and (4) the screen mockups (sketches of the future GUIs). During the project meetings, the industrial partners found the screen mockups (and the glossary) very effective in improving the comprehensibility of the use cases and useful to identify early ambiguities in the requirements specification.

We do not list here the Use Cases, but the services of the system are presented in the next Sections, with information about their implementation and coordination.

The main services are those related to the documents management, and are associated to the GUIs described later. The algorithm for automatically tagging a new document is explained in Section VI. The manual insertion of a new document is described with a concrete example in Section V. Lastly, the system offers services reserved to the administrator of the Node, for example those regarding the approval of the automatically tagged documents, the management of the ontology, users and of the rules for distributing documents over the network: those are cited while describing the Modules that use them.

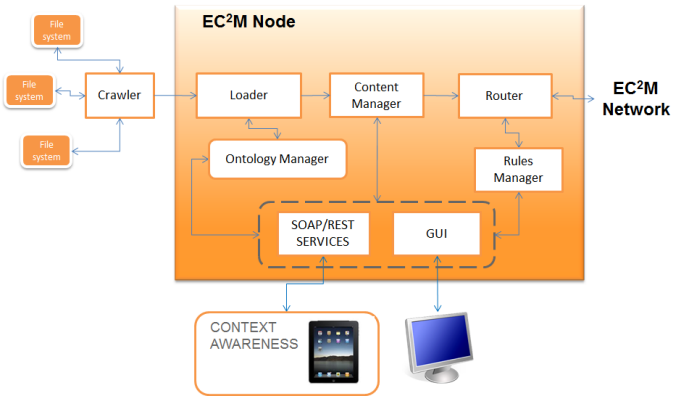


Figure 1. The high level architecture of the EC²M system.

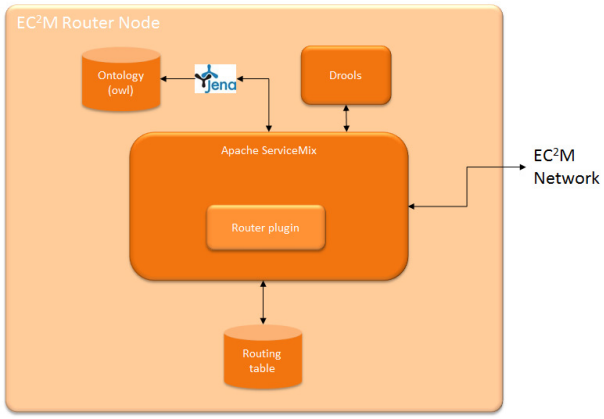


Figure 2. The software architecture of the Central Router.

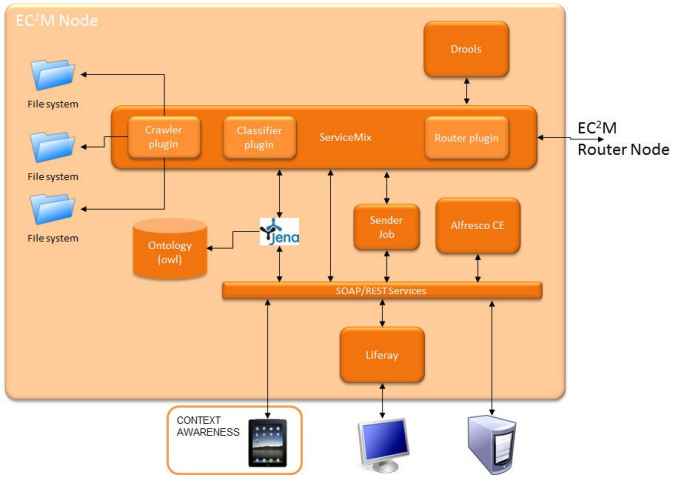


Figure 3. The software architecture of the EC²M system.

Looking at the system architecture from a high level point of view, the system is divided into different modules (see Figure 1):

- Graphical User Interface (GUI): front end of the system, where users can log in from a local terminal and can insert/retrieve/manage documents (based on the ontology);
- Soap and Rest Services: used by the GUIs and by the mobile application which, exploiting context information, lets the user access the system from a remote device;
- Crawler: software that explores predefined hard disk sectors to find documents that are sent to the Loader;
- Loader (or Classifier): module that automatically extracts from the documents tags corresponding to those in the ontology and that enriches the documents with tags related to those already manually associated with the document, using the instances in the ontology and their relationships;
- Ontology Manager: module that queries and manages the ontology;
- Content Manager: module that stores the documents and manages their sharing and retrieval;
- (Local) Router: module that manages the sharing of documents between nodes;
- Rules manager: module that the Router uses in order to define, and dynamically create, routing rules.

On the right side of Figure 1, the arrow points to the EC²M Central Router, reported in Figure 2. In Figure 3, the reader can see which (existing or new) software modules have been used to implement the EC²M system: in the next subsections we give more details on the functionalities and implementation of the different modules.

On each Node Apache ServiceMix is installed: the Crawler, the Classifier and the Router Modules exploit many of its services to implement their functionalities, as described later.

A. Graphical User Interface (GUI)

At design time, we decided to create GUIs simply changeable with respect to the user desires, so that any user can adapt its interfaces. To achieve this goal, we adopted Liferay [26], which is a well known Portal Server. With this platform, the GUIs are portlets that can be plugged into any pre-existing portal. To develop these portlets we used AJAX Vaadin, which is already integrated in the version 6 of Liferay.

Starting from the Use Cases describing the services that must be provided by the system, many GUIs (portlets) have been implemented. The main ones are:

- UploadPortlet: this GUI lets the user insert a new document. As shown later in Section V, from this GUI the user can manually associate tags, selected from the ontology, to the document. Furthermore, as last step before confirming the insertion, the system will suggest the user some more tags to be related to the document;
- SearchPortlet: this GUI lets the user search for documents. This GUI helps the user specifying the search criteria, which will be based on the ontology and on standard document properties (author, creation date and so on);

- ApprovePortlet: users can also put documents in a shared folder and let the Classifier automatically classify them (that is, adding tags from the ontology). These documents must be analyzed by a super user before they are really inserted into the system and shared among users. From this GUI the super user can analyze the automatically associated tags (with the algorithm shown in Section VI), can modify them and lastly can move on with the insertion of the documents in the system;
- SubscribePortlet: from this GUI the user can subscribe to a set of arguments that he is interested in. These arguments are chosen from those in the shared ontology, so that when a new document tagged with one of these arguments is inserted in the network, the user can be automatically informed by the Router Module.

Other GUIs exist to manage the administrative data, such as the users list, the current used ontology, the users permissions and so on.

GUIs exploit the SOAP/REST interfaces, described later, to invoke the services offered by the system.

B. The Crawler and the Classifier Modules

Apache ServiceMix offers many services and components: some of them help polling the file system, also using the FTP protocol. The Crawler Module is a plugin of the local instance of ServiceMix and exploits these services to cyclically search for new documents in predefined shared folders, where users can put documents.

These documents are then sent to the Classifier, which is again a plugin of ServiceMix (but is installed on the cloud): for each document received from the Crawler, it automatically selects related tags (as shown in Section VI) from the ontology and associates them to the document. Then these documents can be analyzed by the superuser to be definitively inserted into the Content Manager.

C. Ontology Manager

To create and manage the ontology we adopted the Web Ontology Language (OWL [27]) and Protégé [28]. The ontology is not subject to frequent changes: if it needs to be modified, this is done using Protégé and then the new version is again made available to the framework for queries.

D. SOAP/REST service

To let different clients interact with the system (from outside of the local Node or from the Node itself), we created a SOAP and a REST interfaces for the main services:

- uploadDocument: it is invoked to add a new document, with its tags, to the content manager;
- search: it lets the invoker searching for a document, specifying as input a list of tags;
- getDocument: it returns the document with the identification number given in input;
- getOntology: it returns the ontology used in the system;
- updatePosition: it is used to send the current position of a mobile device to the Node;
- getTicket: it takes as inputs a username and a password and returns an "authorization token" to let the user be identified in the system;
- subscribeUserFeed: it stores the user interests, given as input;
- retrieveUserFeed: it returns the list of user interests.

E. Content Manager

To physically manage the documents, we adopted Alfresco, a well known open source Java Content Management System: this system allowed us to exploit all the facilities of a high performances business platform with the good property of being an open source software. Furthermore, Alfresco takes advantage of many other well known open source systems like Spring, Hibernate, Lucene and MyFaces, helping developers in creating high quality software in a cost and time limited way.

F. Router

The Router Module is a plugin of ServiceMix, and manages the distribution of the documents in and out of the Node. There is a Router Module on each Node: this module is in charge of informing the node' users of the presence of new documents (local documents or on other nodes) and then of informing the other nodes (thank to the Central Router) of the existence of new documents. The local Router knows the users interested in the new documents (considering the subscribed interests of each user) and, thanks to the Rules Manager Module (described later), can inform them of their existence. Furthermore, it knows the documents exchange rules so it will also distribute the document (or only some parts, for example the title and abstract) to the Central Router (that will spread it to the other interested nodes). The local Router receives the documents from the Central Router too: then it sends these new documents to its Classifier so that they can be inserted into the local CMS.

G. Rules manager

Drools [29] is a well known open source system to create and manage business policies. Being this tool really stable, powerful and already integrated into ServiceMix, we decided to exploit it instead of developing a new module from scratch. The Drools rules are created thanks to an Eclipse plugin and then are read by the Router Module.

For example these are some of the rules regulating the documents notification to users:

- A Notification regarding a new document is sent only to users that subscribed to a topic that appears among the document tags.
- If a requested document is bigger than 1MB, only a link to the document is sent to the Mobile Users (not physically present in the Node), while the complete document is sent if it is smaller than 1MB.

- When a new document is inserted on a Node, only a link to it is to be sent to all the interested local standard (not-mobile) users (because they can directly download it from the local repository).

H. The central Router

The central Route, described in Figure 2, is developed exploiting again Apache ServiceMix as platform, whit the same plugin used for the Router Module, since their functionality is quite the same. As for the other modules, Drools is used to manage the routing rules and Jena is used to interface with the ontology. This Node manages the interaction among the Nodes in the network, distributing the new documents with respects to the subscribed interests of the Nodes. It owns the list of the Nodes in the network, and knows for each Node which are the topics, chosen from the common ontology, they are interested in (that are those the users of that Node are interested in): in this way the Central Router is able to correctly forward new documents among Nodes, avoiding flooding them with useless documents.

Its availability is assured by the cloud platform where it is deployed, reducing the risk of a single point of failure and of low performances.

I. Context Awareness

A mobile application for smartphones was developed for the system: it is able to identify the location of a user currently away from its company node and to act consequently, to let the user be informed of new documents or to let him search for documents in the system from a mobile device.

A "Fingerprint" (a set of information characterizing the Node) was calculated once for each Node and saved in the mobile application, so that it is able to identify when the user is near to its company Node and to move on with the automatic check-in (precondition to use the system). This context awareness algorithm for automatic check-in is called LRACI and is described in details in [30]: its performances are device independent, it is based on GPS/HPS information and is able to exploit Wi-Fi access points in an opportunistic way.

Thanks to this module, that interacts with the system using to the REST/SOAP interfaces developed for it, users can exploit the EC^{2}M system services from mobile devices in a transparent and automatic way.

IV. The Ontology

The EC^{2}M system was designed to work with any ontology describing the documents to be shared and their contents. To start with a concrete example, we decided to design, implement and use an ontology modeling the Sempla's business proposal. With respect to [1], the ontology structure has been changed a little, while some more instances have been created. Furthermore, we added many labels that are used in the automatic classification algorithm and in the GUIs, as described in the next Sections.

Sempla, as a brand, was founded in 2009 and operates in System Integration and Information Technology consulting. It has nearly three decades of experience with the most important Italian groups from the Financial, Production and Public sectors. Now Sempla has been transformed into GFT Italy, member of the GFT Group: we will anyway refer in the paper to Sempla, to keep consistence with [1].

This domain was chosen because Sempla is a very large enterprise, covering different business areas, so its documentation presents many types of documents and a large set of terms that are of interest for different users. These terms and types of documents are quite common in this business area, so modeling the Sempla domain is the best choice because the emerging ontology is correct also for Nacon (that now is member of the Sempla Group) and for Nis (that often collaborates with Nacon so can easily adhere to the ontology), which are the other Nodes in the system.

A. The Ontology Design

To model the domain with an ontology (as defined in [31]), we adopted the Noy and McGuinness methodology [32], which being an agile method is very suitable for collaborating with industrial partners. This methodology foresees these stages:

1) determine the domain and scope of the ontology;
2) consider reusing existing ontologies;
3) enumerate important terms in the ontology;
4) define the classes and the class hierarchy;
5) define the properties of classes-slots;
6) define the facets of the slots;
7) create instances.

The first step was quite simple to follow: the domain of the ontology was the Sempla's business proposal. It is translated into a complex organization of the logic concepts that describe what Sempla offers to its costumers, in terms of products and high technical and management consultancy.

Documents must be tagged to describe, with instances found in the ontology, their structure (some type of documents can have many attachments) and above all their technical and business items: for every business market Sempla has a "portofolio" of products and consultancy services that is well organized and defined.

We searched for similar ontologies, but we were not able to find one that was useful for modeling our domain. Maybe other companies own a similar ontology, but they are not public. We also considered existing ontologies, for example Bibo ontology [33], but we did not use them because they share only very few terms with those used in our domain. We could use the already exiting Sempla's documentation, which offered us an already well-defined set of terms describing the domain, although in natural language and not structured in any standard format.

The third step was conducted with the collaboration of the domain experts, which were the scientists from Sempla, Nacon and Nis. The majority of the terms was collected analyzing the brochures describing Sempla's business proposal (one is summarized in Table I) as well as a large set of documents selected as example from the real ones and a list of terms created by the "users-to-be" of the system, which listed the

TABLE I. THE SEMPLA BUSINESS PROPOSAL (BUSINESS ITEMS) FOR THE "FINANCIAL SERVICES" MARKET, GROUPED BY BUSINESS AREAS.

Business IT Consulting	Digital Marketing & Design	Business Solution	IT Solutions	IT Services	BPO
BPR; Studi di fattibilitá; Enterprise architecture planning; Program management consulting; Organizzazione processi IT; Project portfolio management.	Web & Content Design; User Experience; Community management; Digital Advertising; Augmented Experience.	Credit & Risk Management; Credito Lab; Credito al consumo; Filiale a CRM; Contact center; Pagamenti, Monetica, ATM/POS; Finance & Wealth Planning; Controlli e compliance; Sicurezza e Antifrode; Tesoreria; Human Resources; Reporting & Business Intelligence; Leasing & Factoring; Banca Virtuale; Project Portfolio; General ledger.	System Integration Framework; Application Frameworks; Metodologie di Delivery; Multicanalitá; Enterprise Content Management; DB Administrator.	Application Management; Application Modernization; IT Infrastructure Management; ITIL Implementation.	Contact Centre; Back Office; Fiscalitá Locale; Postalizzazione; Business travel management; Formazione, RollOut, Help Desk.

terms (corresponding to logic concepts used in their work) that they would like to use to classify a document.

To define the classes and their structure, we asked the domain experts to describe in details the types of documents they use, the most relevant information characterizing them and how they model the different business markets. We also took inspiration from the file system where documents were stored: the directories were partially organized as the business areas, and this organization reflected the way Sempla divides its business proposal.

The definition of the properties was done following a similar process.

As a last step, we manually inserted the instances in the ontology: the instances are named using some of the terms listed before, and the properties join them to completely describe the domain. It should be noted that in this process, stating what had to become a class and what an instance was a complex task, because some logical concepts of the domain may be mapped into a class (if we foresee a possible future extension) or into an instance as well (because they are something already stable and with different properties values), for example for *Market* instances. In these cases, we must consider that only instances will be used to tag documents, so it was an obliged decision to model those terms as instances.

Since the ontology is aimed at being used by Italian users, it was created in Italian but some terms (in particular the instances' names) are in English, to maintain a link with the existing documentation and Sempla's internal standards: we are aware that having an ontology with both Italian and English terms was not a perfect solution, but since Classes and Instances should be used as tags for many already existing documents, and considering that their names should be searched inside the text of the documents, we adopted anyway this solution.

B. The ontology details

The Sempla business proposal is organized considering different business markets, to propose ad hoc solutions for each area. Each high level business market is called *Mercato* (Market), and each *Mercato* is characterized by some distinct *Settore* (Sectors) (see Figure 4). To give and explanation of what we assume to be a market, consider that its instances are "Product", "Financial Services" and "Insurance", describing the main activities of the costumers operating in each market. Starting from this macro division of areas, all the other classes are related and organized considering these three sectors. For example, each costumer (class *Cliente_o_Prospect*) refers to one sector, so that his market is uniquely identified.

The Sempla business proposal is created combining different items, identified with the class *Business_Item*, which are divided into different types (subclasses) and that refer to six different Business Areas (as shown in Table I), modeled with the class *Area_Business*, with instances: "Business_IT_consulting", "Business_project_outsourcing (BPO)", "Digital_Marketing_And_Design", "Business_Solution", "IT_services", "IT_solutions".

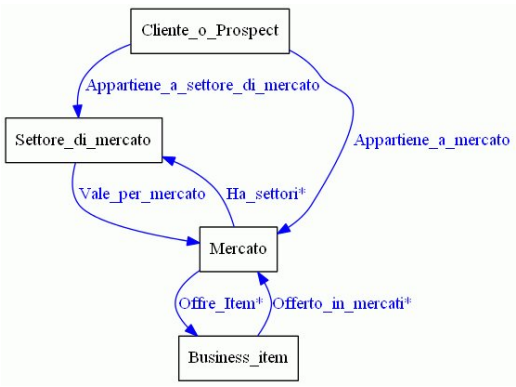


Figure 4. Class *Mercato (Market)* and its main relationships.

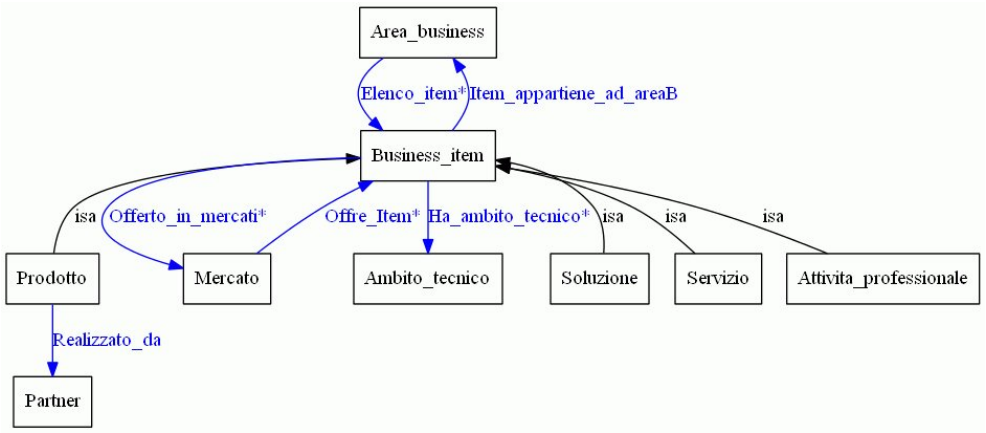


Figure 5. Class *Business_Item* and its main relationships.

Every *Business_Item* can be associated with only one *Area_business* but can be offered to different Markets. A business item that is offered in many markets is called "cross market". The class that groups these items is defined with a necessary and sufficient condition, and is called *BI_Cross_Mercato*. In a similar way, the class *AB_cross_mercato* is defined with the necessary and sufficient condition that it collects the business areas offering at least one "cross market" item.

The business items are divided into four disjoint subclasses: *Prodotto* (Product), which can be developed by a *Partner*, *Soluzione* (Solution), *Servizio* (Service), *Attivitá_professionale* (Professional Activity). These classes are related with class *Ambito_Tecnico* (Technical Area) describing at high level the IT area they refer to (for example "Client Server Application", "Mobile Application" and so on).

The relationships between the classes described above are shown in Figure 5.

The documents that must be classified are divided into different types, modeled with the classes (subclasses of *Documento* (Document)):

- Proposal (class *Documento_offerta*), which describes a business proposal to a costumer;
- Technical attachment (class *Allegato_tecnico_offerta*), which is a technical attachment describing at least one *Business_item* and one *Ambito_Funzionale*;
- Presentation (class *Presentazione*), that describes the Sempla business proposal to a specific Market, and that must be related to at least one *Business_item*.

Class *Ambito_Funzionale*, with its three subclasses, represents a further classification of the business services from a commercial viewpoint. With respect to [1], the subclasses of *Documento* have been slightly simplified.

In Figure 6, a detailed view of the relationships among *Proposal* and the other classes is shown: class *Proposal* has one string property ("Id_Proposal") not shown in Figure 6, which uniquely identifies the proposal. In Figure 7, an overview of the relationships among the different types of documents is reported.

Class *Partners* represents a list of possible third parties companies involved in the proposal.

We only show Object properties in figures and do not report instances of every class. In Section V, the reader can find some of these instances shown in figures and tables, while they are used by the EC^{2}M interface to help user tagging a document.

C. Exploiting the ontology

As described before, the main goal of the ontology is to formally model the domain and to store the common information needed to tag documents, using the ontology instances, in the system.

But the ontology has been also used to store other information related to Classes and Instances that will be used by the GUIs and by the automatic classification module.

Regarding the exploitation of the ontology by the GUIs, please consider that the EC^{2}M system was created to help different users, from different companies, to share documents. It is possible that users speak different languages: consequently, the GUIs should adapt to the desired language. Similarly, some classes or instances names suffer of a formalism that is due to OWL, but that could create confusion to the end users. Since GUIs read from the ontology the possible tags and have to show the users them plus the ontology class the tags refer

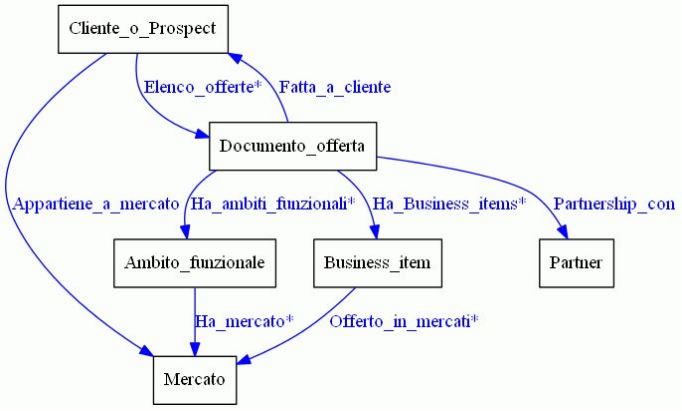


Figure 6. Class *Documento_offerta* (*Proposal*) and its main relationships.

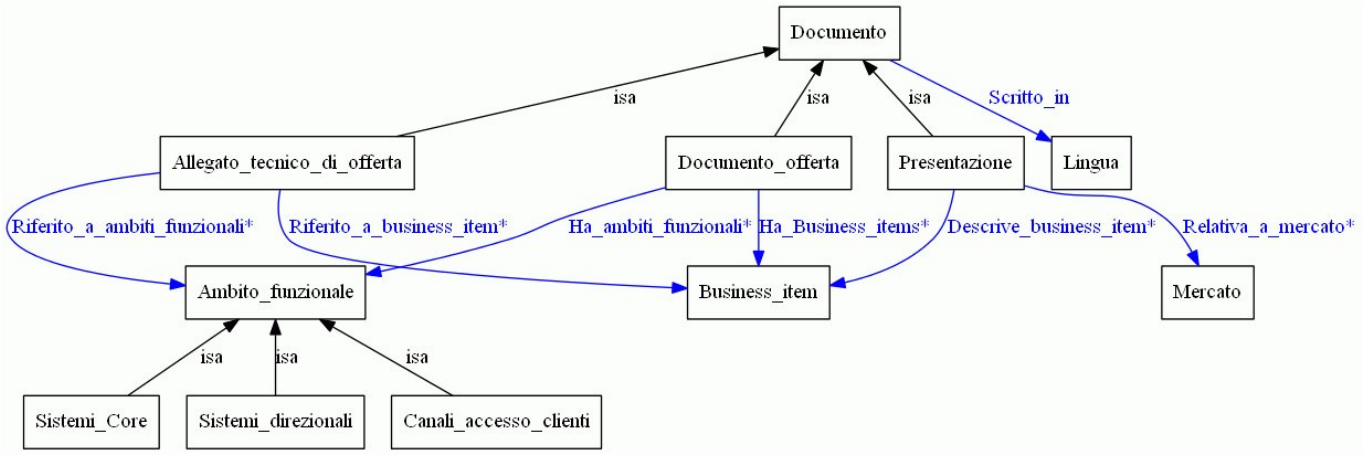


Figure 7. Relationships among classes starting from *Documento* (*Document*) and its subclasses.

to, we added to ontology classes some rdfs:labels to store the value that must be visualized in the GUIs. For example, as shown in Section V and Section VI, to Italian users the class *Business_Item* is shown with the name "Elemento d'offerta", read from the label associated to that class.

Regarding the exploitation of the ontology by the automatic classification module, we improved the information stored in the ontology by adding more synonyms, acronyms, equivalent definitions and generic information to classes and instances, storing them into rdfs:labels: these terms are the elements of the "synsets" that will be read and used by the automatic classification module, as shown in Section VI.

V. THE RUNNING SYSTEM

A first prototype of the EC^{2}M system has been developed at the end of 2012, based on the ontology described in Section IV of [1], and has been tested concretely by Nacon, Nis and Sempla to manage the documents created in 2012 (3200 documents), with 31 users. Now the system is completely updated and running (using the new version of the ontology presented in this paper), serving more than 600 users, and manages a documents set that grows each year approximatively of 3000 new items.

Some images of the GUIs are reported to give an example of usage. The GUI form in Figure 8 is the one where the user can add a new document, specifying: what kind of document he is inserting (a "Presentazione"), the market (in this case "Insurance") and the business items the document is about (selecting these values using drop down lists that group the business item instances in their subclasses). In the example, the user chooses the instances "User_Experience", "Web_development" and "Digital_media_strategy" from class *Attivitá_professionale* (instances of class *Attivitá_professionale* are shown in Table II). Finally, the user chooses the language ("Italiano").

Figure 8. First GUI form for inserting a new document with some first tags. Screen shot.

In Figure 9, the system presents to the user the list of the current tags that he selected plus the tags that have been automatically added: in this case only the tag "Digital_Marketing_Design" from class *Business_Area*, because all the selected business items refer to this area, as shown in Table III.

Note that in the ontology, labels have been added to classes to store the term to be shown in the GUIs, because sometime the class names are not "good to be visualized" (for example they contain the "_" character, or are in a different language from the GUI one). For example, in Figure 9 the tags from class *Business_Item* are called "Elemento d'offerta" instead of the standard class name.

Then, the system asks the user if he wants to add some more tags, choosing from those connected to the already selected ones. In the example, the user chooses to add more tags starting from the business area "Digital_Marketing_Design". So the system shows the user the possible tags, choosing from the instances related to "Digital_Marketing_Design" (considering the properties with domain *Business_Area*) in the ontology (Figure 10). The user can add some of those tags and then saves the document.

TABLE II. INSTANCES OF THE *Attivitá_professionale* CLASS (SUBCLASS OF *Business_Item*).

User Experience	Web development	Digital media strategy	ADV Campaign
Program management consulting	IT infrastructure management	Project portfolio management	Enterprise architecture planning
ITIL Implementation	Business driven development	Delivery Methodologies	IT Processes Organization
Application Modernization	Digital Marketing	Feasibility Study	Visual Graphic Design
BPR	Brand Identity	Time Material	Movie Design

TAG	CATEGORIA
Insurance	Mercato
User Experience	Elemento d'offerta
Web development	Elemento d'offerta
Digital media strategy	Elemento d'offerta
Italiano	Lingua
Digital_Marketing_Design	Area_business

Figure 9. List of a manually selected tags plus those automatically added. Screen shot.

TABLE III. *Business_Item* INSTANCES ASSOCIATED TO THE "DIGITAL_MARKETING_DESIGN" BUSINESS AREA.

User experience	Movie Design	ADV Campaign	Brand Identity
Digital Media Strategy	Visual Graphic Design	Digital Marketing	Web Development

Scegli ulteriori tag da associare al documento

TAG	CATEGORIA
Community	Ambito tecnico
Forum	Ambito tecnico
Portale	Ambito tecnico
Sito web	Ambito tecnico
Sito mobile	Ambito tecnico
Product	Mercato
Financial Services	Mercato

TAG	CATEGORIA
Insurance	Mercato
User Experience	Elemento d'offerta
Web development	Elemento d'offerta
Digital media strategy	Elemento d'offerta
Italiano	Lingua
Digital_Marketing_Design	Area_business

Figure 10. Possible new tags (left) and already associated tags (right). Screen shot.

VI. THE AUTOMATIC CLASSIFICATION ALGORITHM

The EC^{2}M system is equipped with the Loader/Classifier module, able to automatically classify a document given as input: it is based on Lucene [34] (to interact with Alfresco for getting the indexes and to analyze the documents, searching for terms) and on Jena [35] (to read the ontology classes, instances and labels). It follows an algorithm based on [36], and is used to automatically identify the type of the input document and the possible related tags (chosen from the ontology). These automatically associated tags are then presented to the user who can confirm or change them.

The UML activity diagram in Figure 11 (created following the methodology described in [37], as for that in Figure 12) shows the Classifier's algorithm devoted to the identification of the document type:

- It asks the Ontology Manager (that uses Jena) the list of Document Types (that are the subclasses of the ontology class *Documento* (Document)) with their related synsets (that are the sets of rdfs:labels associated to each class);
- For each document type, it:
  - searches in the document title the terms in the document type synset;
  - searches in the document body the terms in the document type synset;
  - If at least one correspondence is found, the type is considered in the list of possible candidates, and a score is calculated based on the frequency of the synset terms found in the previous steps. If the term is found in the title, it is multiplied for a predefined "boost value".
- At the end of the loop, the document type with the highest score (that is, the uniquely found document type or that with a score exceeding of a predefined value, called "DELTA", those of the other document types) is associated to the document. If there are many types with similar scores, none type is associated to the document.

The algorithm that automatically extracts the semantic tags is described with an activity diagram in Figure 12:

- It asks the Ontology Manager (that uses Jena) the properties list of the document type chosen in the previous phase of the algorithm (if any);
- For each datatype property, it calculates the possible value analyzing with Lucene the document, searching for the property name (or for the terms in its synset, stored again as rdfs:labels related to the property) followed by a value of the correct property type. This couple is added to the set of indexes related to the document (indexes are "not semantic" standard tags that can be related to a document in a CMS: they are managed directly by Alfresco apart those created in this step, which are automatically calculated as described and added to this set);
- For each Object Property, it asks the Ontology Manager the instances of that Class with their related synsets (that are the sets of rdfs:labels associated to each instance) and for each instance:
  - searches in the document title the terms in the synset;
  - searches in the document body the terms in the synset;
  - If at least one correspondence is found, the instance is considered in the list of possible candidates, and a score is calculated based on the frequency of the synset terms found in the previous steps. If the term is found in the title, it is multiplied for a predefined "boost value";
- At the end of the loop, if the property has single cardinality, the instance with the highest score (that is, the uniquely found instance or the instance with a score exceeding of a predefined value, called "DELTA", those of the other instances) is associated to the document while, if it has multiple cardinality, all the instances with a score higher than a predefined "threshold" are associated to the document;
- Lastly, considering each instance associated to the document in the previous steps, its data properties with a value are added as indexes, and other instances (if any) related by object properties are added as tags to the document too.

The "boost value" is a parameter used to give some more importance (that, in this case, is an higher score) to a term if it has been found in the title of a document. Its value may range between 1 and 2. After many tests to identify the most

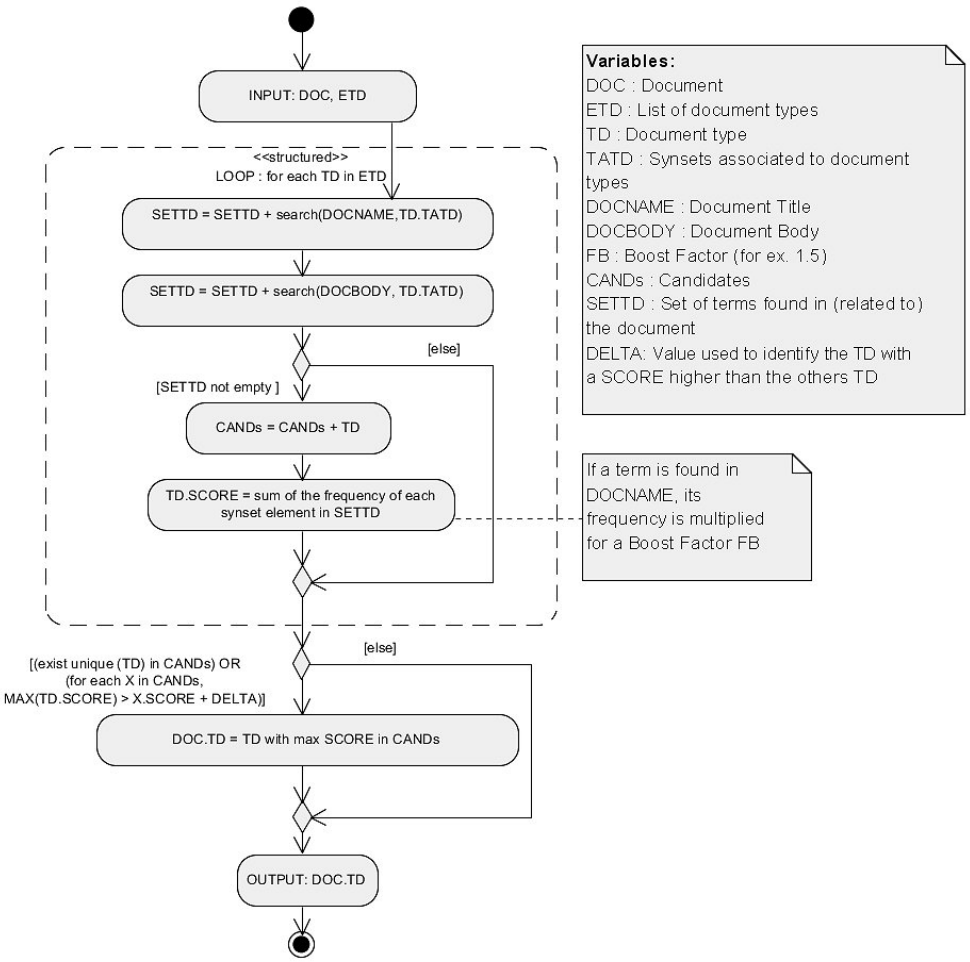


Figure 11. Activity Diagram describing the algorithm to identify the document type.

correct value for this parameter, we now use it with value 1.5. In general, if you have a domain with documents with relevant titles, an higher value of this parameter is recommended.

The "DELTA" parameter, as explained before, is used to select only the instance with a score really higher then that of the other candidates, and again has been tuned during the tests: its value may range between 0 and 1, and we use it set to 0.2.

Lastly, the "threshold" parameter is used to select only a subset of the identified candidates to be proposed to the user: at the end of the algorithm we have a list of candidates with a score, which ranges from 0.1 (the lowest value) to 1 (the highest one), reflecting the confidence that the candidate is really associable to the document. We use the "threshold" parameter to choose, between them, only those with a reliable score. Its value may range between 0 and 1, and currently this value is set to 0.3 (as explained later in this Section).

An example is reported in Figure 13: the document in input, "06-Portale del Credito-CR2 - Revisione PDC GOR - MS.pptx", is automatically correctly identified as a *Presentazione* (Presentation).

This class has two properties, as shown in Figure 7: "Scritto_In (Written In)" (single, with range *Lingua* (Language)), and "Descrive_Business_Item (visualized as "Describe" in Figure 13)" (multiple, with range *Business_Item*). The algorithm correctly identifies the language (instance "Italian") and selects three related *Business_Item* for the property "Describe" ("Credito al consumo", "Credito lab", "Credit and risk management").

As last step, considering that all the three *Business_Items* are related to the "Business Solution" instance of *Business_Area* class (see Table I) thanks to the property "Item_appartiene_ad_areaB" (see Figure 5), the tag "Business Solution" is associated to the document.

The first tests, presented in [1], gave already promising results: considering the automatic tags extraction, those with a threshold higher than 0.3 were correctly associated with the document in the 95% of the executed tests. In the following months the system has been tuned to achieve better results: the algorithm was not modified, while the ontology has been changed to create more complete synsets (in fact many wrong tags associations were due to poor synsets). Now that the ontology has been updated and revised, the percentage of correct tags is 100% with threshold higher than 0.5, and 97% with threshold between 0.3 and 0.5.

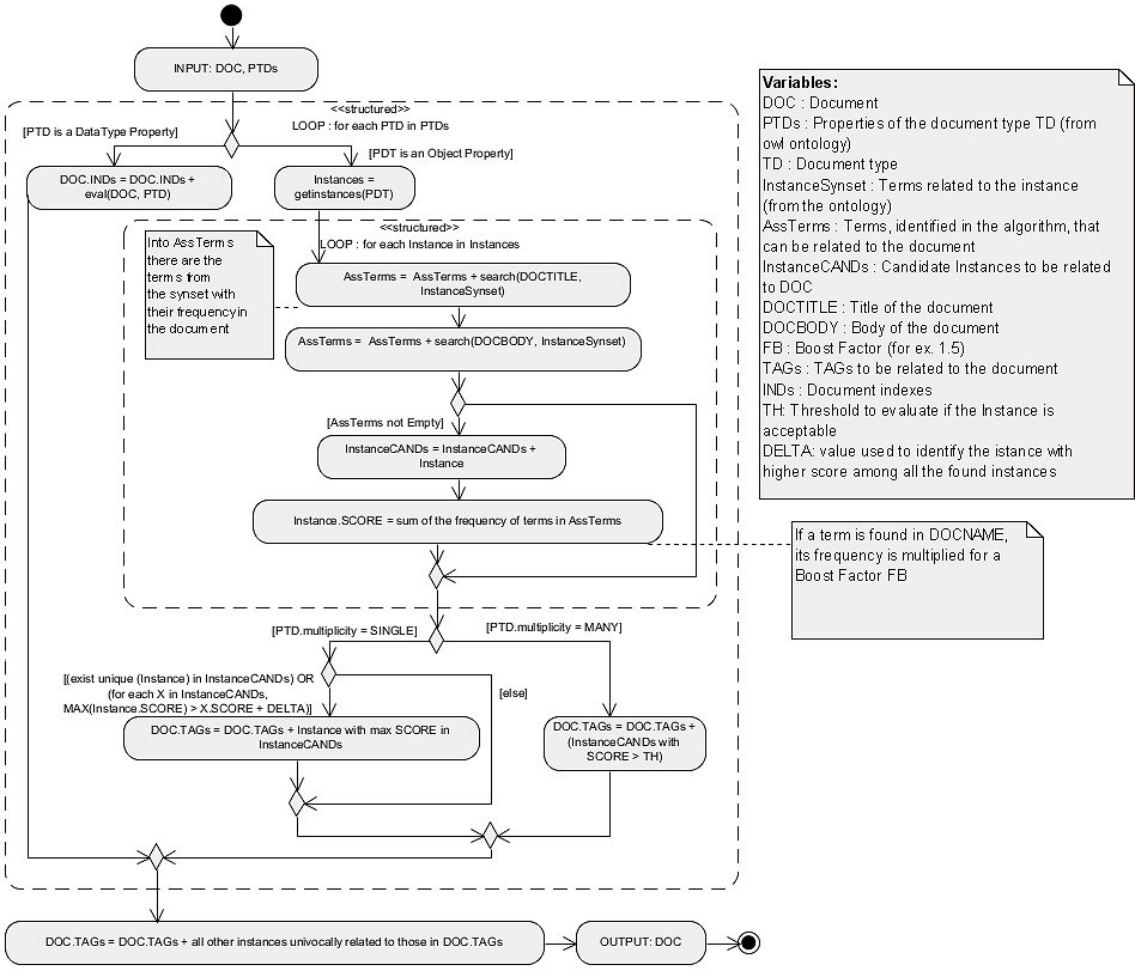


Figure 12. Activity Diagram describing the algorithm to identify the document tags.

Document type	Score
Presentation	1.0
Proposal	0.2
Technical Attachment	0.1

Property	Cardinality	Class	Instance	Score
Describe	Multiple			
		Product	Credito_lab	1.0
		Solution	Credit e risk management	0.83
		Solution	Credito al consumo	0.5
		Time material	Organizzazione_processi_IT	0.07
		Solution	Gestione sinistri e reporting	0.05
		Time material	Web development	0.03
		Solution	Controlli e compliance	0.03
		Solution	Reporting e business intelligence	0.03
		Time material	Digital media strategy	0.02
		Service	Application Management	0.01
		Time material	Studio fattibilità	0.01
		Time material	Web Content Management	0.01
		Solution	Business travel management	0.01
Written in	Single			
		Language	Italian	1.0

Class	Instance
Business area	Business solution
Solution	Credit e risk management
Language	Italian
Solution	Credito al consumo
Product	Credito_lab

Figure 13. Automatically identified tags (highlighted rows) in the document used for example.

VII. Conclusion and Future Work

The solution that we adopted is based on a mesh-up of many different technologies and softwares, where an ad hoc ontology is integrated into an open-source CMS and then deployed over a cloud platform, following the new trends in different research areas. As shown in Sections I and II, many solutions exist that are in some way similar to ours (considering the adoption of a formal representation of the domain, or for the similar overall structure) but are really far for other aspects (different domain, different semantic representation and expressiveness, or simply being only a study, not a real and applied tool). It is impossible to make a precise comparison with the other mentioned systems, because they are commercial or not available to the community, or because a completely new system, using the other approaches or technologies, should be developed to be tested and compared, and this solution is not feasible. As far as we were able to do, we adopted the open-source softwares recognized by the users and by the research community as very reliable and efficient ones (for example Lucene and Alfresco), while our industrial partner chose the cloud platform making again a detailed analysis of the available platforms (*Eucalyptus* and *Amazon Web Services* by Amazon, *OpenStack* by Rackspace, *AppScale* by Google), to adopt the best one for our project. To develop our ontology, considering that it was the core of the system, we adopted OWL, since it is one of the most expressive language for developing ontology and that some well working libraries to interface and manipulate it from an external program exist (like Jena).

The EC^{2}M project has been fully tested in 2012 and 2013, and showed very good results with respect to a standard ECM system: it really helped users from different companies better collaborate, exploiting a semantic classification of their documentation and consequently offering a simpler searching phase and a better support in sharing information, which was impossible to obtain without a similar system.

The ontology is now complete and models all the document types and contents. It has been equipped with labels to store both "visualization information" (that are the labels that must be visualized in the GUIs) both "synsets". This solution was definitely good for our system because:

- it lets the system administrators simply modify some parts of the GUIs only modifying the ontology, reducing the chance of an misalignment between the terminology and the GUIs, and reducing the time-to-market too
- it lets simply modify the instances and their synsets, so that the automatic classification algorithm can consequently improve its results thanks to a more complete ontology

keeping the independence between the domain representation, the GUIs and the automatic classification algorithm itself.

The notification of new documents to subscribed users is done in quasi real time, as requested by these types of applications, for the remote and mobile users too. Furthermore, with the deployment over the cloud platform, the performances can be enhanced with a new purchase of cloud services: with this architecture and deployment solution the system is really scalable.

The system has been adopted by GFT (previously Sempla), and related companies, as their new content management system, since it showed very good performances (thanks to the cloud architecture), great accessibility (thanks to the mobile access and context awareness module), high reliability in the automatic tagging process, and high adaptability (thanks to the adoption of the ontology as an independent domain representation). So, it is not only a prototype, but a real and running system that shows how these solutions coming from the research fields (ontologies, context awareness, cloud and CMS composed together) can actually and effectively work in real industrial scenarios.

Furthermore, the adoption of open source technologies and the chance of exploiting the system as a service over the cloud platform (with a pay-per-use solution) allowed a reasonable initial budget and a high scalability in the overall architecture, which from the industrial viewpoint is a good "Return Of Investment": that is another evidence of the applicability of these technologies in industrial systems.

As last future work, we will investigate how dealing with the scenarios where the nodes in the EC^{2}M network use different ontologies to describe and tag their documentation: in this case, the common ontology must be anyway chosen and defined, but ontology matching techniques may be adopted to align the common and private ontologies before tagging and when receiving notifications from the other nodes, to let them keep on using their private ontology but also being able to share documents with common tags.

Acknowledgment

The research described in this paper has been funded by the "EC^{2}M system (Enterprise Cloud Content Management)" Programma Operativo Regionale (POR) project.

The authors would like to thank Maurizio Ferraris from Nacon and Viviana Mascardi and Gianna Reggio from DIBRIS that led the industrial and the DIBRIS academic components, respectively, involved in the EC^{2}M project. Thanks also to Maurizio Leotta for the activity Diagrams shown in this article.

A special thanks goes to Dante Laudisa, from Sempla, who actively participated in the domain explanation and in the ontology definition.

References

[1] D. Briola, A. Amicone, and D. Laudisa, "Ontologies in Industrial Enterprise Content Management Systems: the EC^{2}M Project," in COGNITIVE 2013, The Fifth International Conference on Advanced Cognitive Technologies and Applications, 2013, pp. 153–160.

[2] Association for Information and Image Management, "What is Enterprise Content Management (ECM)?" 2010, URL: http://www.aiim.org/what-is-ecm-enterprise-content-management.

[3] A. P. Sheth and C. Ramakrishnan, "Semantic (Web) Technology In Action: Ontology Driven Information Systems for Search, Integration and Analysis," IEEE Data Eng. Bull., vol. 26, no. 4, 2003, pp. 40–48. [Online]. Available: http://dblp.uni-trier.de/db/journals/debu/debu26.html#ShethR03

[4] R. Andersen, "The Rhetoric of Enterprise Content Management (ECM): Confronting the Assumptions Driving ECM Adoption and Transforming Technical Communication," Technical Communication Quarterly, vol. 17, no. 1, 2008, pp. 61–87.

[5] "The Alfresco Homepage," 2014, URL: http://www.alfresco.com/ [accessed: 2014-03-01].

[6] "The Plone Homepage," 2014, URL: http://plone.org/ [accessed: 2014-03-01].

[7] "The Sensenet Homepage," 2014, URL: http://www.sensenet.com/ [accessed: 2014-03-01].

[8] "The smartlogic Homepage," 2014, URL: http://www.smartlogic.com [accessed: 2014-03-01].

[9] "The H-Dose Homepage," 2014, URL: http://dose.sourceforge.net/ [accessed: 2014-03-01].

[10] "The OpenCalais Homepage," 2014, URL: http://www.opencalais.com/ [accessed: 2014-03-01].

[11] "The Apache Stanbol Homepage," 2014, URL: http://stanbol.apache.org/ [accessed: 2014-03-01].

[12] R. García, J. M. Gimeno, F. Perdrix, R. Gil, and M. Oliva, "The Rhizomer Semantic Content Management System," in WSKS (1), ser. Lecture Notes in Computer Science, M. D. Lytras, J. M. Carroll, E. Damiani, and R. D. Tennyson, Eds., vol. 5288. Springer, 2008, pp. 385–394. [Online]. Available: http://dblp.uni-trier.de/db/conf/wsks/wsks2008.html#GarciaGPGO08

[13] C. Frank and M. Gardoni, "Information Content Management with Shared Ontologies-at Corporate Research Centre of EADS," Int. J. Inf. Manag., vol. 25, no. 1, Feb. 2005, pp. 55–70. [Online]. Available: http://dx.doi.org/10.1016/j.ijinfomgt.2004.10.009

[14] D. M. Le and L. M. S. Lau, "An Open Architecture for Ontology-Enabled Content Management Systems: A Case Study in Managing Learning Objects," in OTM Conferences (1), ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., vol. 4275. Springer, 2006, pp. 772–790. [Online]. Available: http://dblp.uni-trier.de/db/conf/otm/otm2006-1.html#LeL06

[15] G. Laleci, G. Aluc, A. Dogac, A. Sinaci, O. Kilic, and F. Tuncer, "A Semantic Backend for Content Management Systems," Knowl.-Based Syst., vol. 23, no. 8, 2010, pp. 832–843. [Online]. Available: http://dblp.uni-trier.de/db/journals/kbs/kbs23.html#LaleciADSKT10

[16] H.-C. Chu, M.-Y. Chen, and Y.-M. Chen, "A Semantic-based Approach to Content Abstraction and Annotation for Content Management," Expert Syst. Appl., vol. 36, no. 2, Mar. 2009, pp. 2360–2376. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2007.12.067

[17] A. Bechini and A. Vetrano, "Management and Storage of in Situ Oceanographic Data: An ECM-based Approach," Inf. Syst., vol. 38, no. 3, May 2013, pp. 351–368. [Online]. Available: http://dx.doi.org/10.1016/j.is.2012.10.004

[18] B. Thönssen, "An Enterprise Ontology Building the Bases for Automatic Metadata Generation," in MTSR, ser. Communications in Computer and Information Science, S. S. Alonso and I. N. Athanasiadis, Eds., vol. 108. Springer, 2010, pp. 195–210. [Online]. Available: http://dblp.uni-trier.de/db/conf/mtsr/mtsr2010.html#Thonssen10

[19] J. Corchado, D. Tapia, and J. Bajo, "A Multi-Agent Architecture for Distributed Services and Applications," Computational Intelligence, vol. 24, 2008, pp. 77–107.

[20] D. Griol, J. M. Molina, and Z. Callejas, "Providing Personalized Internet Services by means of Context-Aware Spoken Dialogue Systems," JAISE, vol. 5, no. 1, 2013, pp. 23–45. [Online]. Available: http://dblp.uni-trier.de/db/journals/jaise/jaise5.html#GriolMC13

[21] G. Blázquez, A. Berlanga, and J. M. Molina, "inContexto: Mobile Phone Multi-Sensor Architecture to Obtain People Context," Journal of Ambient Intelligence and Smart Environments, vol. 5, 2013, pp. 23–45.

[22] R. Fuentes-Fernández, J. Gómez-Sanz, and J. Pavón, "Understanding the Human Context in Requirements Elicitation," Requirements Engineering, vol. 15, no. 3, 2010, pp. 267–283. [Online]. Available: http://dx.doi.org/10.1007/s00766-009-0087-7

[23] J. L. R. García and A. L. Tello, "Ontotv: an Ontology-Based System for the Management of Information about Television Contents," Int. J. Semantic Computing, vol. 6, no. 1, 2012, pp. 111–. [Online]. Available: http://dblp.uni-trier.de/db/journals/ijsc/ijsc6.html#GarciaT12

[24] A. Aiello, M. M. Furnari, A. Massarotti, S. Brandi, V. Caputo, and V. Barone, "An Experimental Ontology Server for an Information Grid Environment," International Journal of Parallel Programming, vol. 34, no. 6, 2006, pp. 489–508. [Online]. Available: http://dblp.uni-trier.de/db/journals/ijpp/ijpp34.html#AielloFMBCB06

[25] G. Reggio, F. Ricca, and M. Leotta, "Improving the Quality and the Comprehension of Requirements: Disciplined Use Cases and Mockups," in Proceedings of the 40^{th} Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2014). IEEE, 2014.

[26] "The Liferay Homepage," 2014, URL: http://www.liferay.com/ [accessed: 2014-03-01].

[27] "The OWL Language Overview Homepage," 2014, URL: http://www.w3.org/TR/owl-features/ [accessed: 2014-03-01].

[28] "The Protégé Homepage," 2014, URL: http://protege.stanford.edu/ [accessed: 2014-03-01].

[29] "The Drools Homepage," 2014, URL: https://www.jboss.org/drools/ [accessed: 2014-03-01].

[30] I. Bisio, F. Lavagetto, M. Marchese, and A. Sciarrone, "GPS/HPS- and Wi-Fi Fingerprint-Based Location Recognition for Check-In Applications Over Smartphones in Cloud-Based LBSs," Multimedia, IEEE Transactions on, vol. 15, no. 4, June 2013, pp. 858–869.

[31] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications," Knowl. Acquis., vol. 5, no. 2, Jun. 1993, pp. 199–220. [Online]. Available: http://dx.doi.org/10.1006/knac.1993.1008

[32] N. F. Noy and D. L. Mcguinness, "Ontology Development 101: A Guide to Creating Your First Ontology," Tech. Rep., 2001.

[33] "The Bibliographic Ontology Homepage," 2014, URL: http://bibliontology.com/specification [accessed: 2014-03-01].

[34] "The Apache Lucene Homepage," 2014, URL: https://lucene.apache.org/ [accessed: 2014-03-01].

[35] "The Jena Homepage," 2014, URL: https://jena.apache.org/index.html [accessed: 2014-03-01].

[36] K. S. Jones, "A Statistical Interpretation of Term Specificity and its Application in Retrieval," Journal of Documentation, vol. 28, 1972, pp. 11–21.

[37] G. Reggio, M. Leotta, F. Ricca, and E. Astesiano, "Business Process Modelling: Five Styles and a Method to Choose the Most Suitable One," in Proceedings of the Second Edition of the International Workshop on Experiences and Empirical Studies in Software Modelling, ser. EESSMod '12. New York, NY, USA: ACM, 2012, pp. 8:1–8:6. [Online]. Available: http://doi.acm.org/10.1145/2424563.2424574

Supporting Ambient Assisting Living by using Executable Context-Adaptive Task Models

Estefanía Serral
Department of Decision Sciences and Information Management
KU Leuven, Belgium
Email:estefania.serralasensio@kuleuven.be

Pedro Valderas and Vicente Pelechano
ProS Research Center
Universitat Politècnica de València
Email:{pvalderas, pele}@pros.upv.es

***Abstract*—The amount of elderly people with chronic diseases is constantly increasing, and current health systems are not able to provide a proper supervision. Ambient Assisted Living (AAL) is a new research area that stands for the use of pervasive and mobile technologies in order to increase the quality of life, wellbeing and safety of elderly people. In this work, we present a tool-supported methodology to facilitate the creation of AAL systems through the use of executable models. AAL services are specified by executable context-adaptive task models by using concepts of a high level of abstraction, which facilitate the participation of medical professionals in the AAL specification. The task models are then interpreted and executed at runtime by a software infrastructure that automates the AAL services as specified. Thus, task models are the only implementation of the services, making it easy their further evolution after system deployment. In order to demonstrate the feasibility of our methodology, we have evaluated it in the development of an AAL system for assisting the patients of a nursing home.**

***Index Terms*—Ambient Assisting Living, smart environments, models at runtime, context adaptation;**

I. Introduction

This paper introduces an evolution of the software infrastructure presented in [1] in order to support the development of Ambient Assisting Living (AAL) environments [2]. The ageing of population is making that the number of citizens with chronic diseases is increasing, especially among elderly people. These people require a constant control and supervision that traditional public health systems are not able to provide in a satisfactory way. The advances in the Internet of Things (IoT), and the progress of mobile devices (smart phones, tablets, etc.) and wearables (Google glasses, smart watches, sensors in clothes and body, etc.) allow creating systems that sense patients' context in order to take decisions that adapt system behaviour according to the patients' needs.

This is the main goal of AAL practices, which stand for the use of electronic processes and communication as well as mobile technologies in order to support the practice of medicine and public health. AAL applications include services, products and concepts to increase the quality of life, wellbeing and safety of elderly people. The main goal of AAL is to achieve benefits for the individual (increasing safety and well-being), the economy (higher effectiveness of limited resources) and the society (better living standards) [3]. A clear evidence of the current interest in these topics is the European Ambient Assisted Living joint Program and all the submitted projects [4].

AAL environments require self-adaptive systems to automate AAL services that assist patients when is required in a non-intrusive way. These systems also need to consider the patients and medical staff (which will be the end-users of the system), in the system design in order to properly support medical guidelines and patients demands. Several works such as [5][6][7][8] have worked on the automation of tasks and system adaptation. However, these solutions focus on the technical challenge of adapting a system by analyzing user behavior and environment conditions at runtime, paying little attention to the involvement of end-users from early stages of development, which is crucial on AAL environments.

To improve these challenges, we evolve the software infrastructure presented in [1] to be applied in AAL environments. This infrastructure was proposed for automating tasks in smart homes. This paper extends it to successfully automate assisting services. In addition, we propose a methodology for using this infrastructure in order to support the development of these services from the requirement elicitation until their execution. In order to properly identify the assisting services required by the end-users, we use User Centered Design (UCD) techniques such as Personas and scenarios [9]. The identified services are then described in context-adaptive task models. Task models facilitate the participation of patients and medical professionals in their definition and allow describing the assisting services in a very intuitive manner by using high-level concepts close to the problem domain.

The tasks models are directly interpreted by a software infrastructure at runtime, which executes the described assisting services in the appropriate context. This allows automating the services from the very moment in which task models are defined. Moreover, this facilitates the further evolution of the services if health conditions of patients change: by only updating the models, the services are evolved. With this infrastructure, we can provide patients with a high quality of assistance. In addition, assisting services can be performed in a convenient way for patients since they are analyzed by medical professionals before they are automated by using the task model. Moreover, assistance is self-adaptive according to context, i.e., the software infrastructure reacts and autonomously adapts patient assistance according to each context. In order

to evaluate our approach we have used it to develop an AAL system that assists the patients of a Nursing Home.

The rest of the paper is organized as follows: Section II presents the related work. Section III introduce the proposed methodology to support assisting services from their requirements to their execution. Section IV presents the process to capture the assisting service requirements. Section V describes the models proposed to specify the needed assisting services. Section VI presents the software infrastructure that automates assisting services in a context adaptive way and allows their evolution after system deployment. Section VII validates the approach using a case study based evaluation. Section VIII concludes the paper and proposes further work.

II. Related Work

An AAL scenario is characterized by being connected, context-adaptive and anticipative. In order to confront the creation of these environments, AAL systems are usually structured in three levels [10]: Hardware (sensing, wireless networks), Middleware (data capture, data safety, IT integration) and Services (biosignal processing, application-orientated processes, community services).

A lot of research work has been done in all three levels. An overview on assisting hardware technology is given in [11]. It addresses video-monitoring, remote health monitoring, electronic sensors, and equipment such as fall detectors and door monitors. This technology alone is not enough to coordinate hardware components in order to provide complex assisting services. Thus, only panic buttons-based solutions can be provided.

To enable increased safety and wellbeing in a specific environment, it has to become intelligent with the help of pervasive devices, which are capable to register changes in the physical environment and thus actively interact in a process. In addition, a system that coordinates these devices is required. In this context, there are a number of research projects focusing on the development of AAL middlewares. For example, the PERSONA project [12] proposes a middleware platform for the implementation of semantic assisting services. The Aware Home [13] project built up a living lab, in which they tested the users acceptance of technology, building up a bridging framework for universal device interoperability in pervasive systems. The mission of I-Living [14] is developing an assisted-living supportive software infrastructure that allows disparate technologies, software components, and wireless devices to work together. Tasks provided in I-Living are such as activity reminding, health monitoring, personal belonging localization, emergency detection, and so on. However, the available services provided by these three projects are closed and still limited, and they do not provide tools that facilitate end-users to participate in the definitions of these services.

According to [15], there are three types of assisting services: (a) Emergency treatment, which faces situations such as sudden falls, heard attacks, strokes, panics, etc.; (b) Autonomy enhancement, which allows replacing medical and social care personnel by appropriate system support such as a cooking assistance system for people with visual defects; and (c) Comfort, which covers all areas that do not fall into the categories (a) and (b) such as social contact assistance, infotainment assistance, logistic assistance, etc.

Independently from their type, AAL services imply the execution of a set of tasks in a coordinated way. For instance, a service that treats a fall may require analyze patient's location and state, and people surround them, and alert doctors or caregivers if a fall emergency is detected; a service for improving autonomy may require to check the fridge for essential products and decide to make a shopping order if needed according to the diet of the users; and a service that manages comfort may require to graduate the light intensity and the temperature, close or open blinds and windows, and play certain music according to the users' taste.

Note that the above presented AAL approaches provide little support to define this type of coordination, where tasks are executed in a specific order depending on environment conditions, users' state, or the outputs of previous tasks. The research fields of task automation and context-awareness play key roles in order to support these three types of services. Machine-learning approaches have attempted to deal with the automation of user routines by automatically inferring them from past user actions [5][6]. These approaches have done excellent work by automatically learning from user behavior; however, assisting services need to be available from the very beginning deployment of the system and a learning process is not acceptable [16]. In addition, these approaches may be intrusive for users because the repeated execution of an action does not imply that the patients or caregivers wants this automation. Also, they reproduce the actions that users have frequently executed in the past and in the same manner that they were executed. This prevents user tasks from being carried out in a more efficient and convenient way, i.e., including medical professional guidelines, and does not allow tasks that users did not perform before to be automated.

Context-aware rule-based approaches have made great advances in introducing context into software systems. To automate user tasks, they program rules that trigger the sequential execution of actions when a certain context event is produced [7][8]. However, these techniques are only appropriate for automating relatively simple tasks [17]; hence, they usually require large numbers of rules. If we also consider that these rules have to be manually programmed [17], the understanding and maintenance of the system may become very difficult.

In this work, we propose a solution based on executable task models. The concept of task is intuitive enough to be understandable by medical professionals and persons resposible of the patients, facilitating their involvement in the development process; they provide rich expressiveness that allows us to precisely describe the assisting services that the system must support to face specific situations; and they can be automatically interpreted by a software infrastructure from the initial deployment of the system. Historically, task-oriented computing uses task modelling to facilitate the interaction of users with the system. These systems have proven that

task modelling is effective in several fields such as user interface modelling [18], assisting end-users in the execution of tasks [19], etc. These works show the growing usage of task modelling and its remarkable results and possibilities to model system behavior. However, none of these works attempt to describe AAL services. Hence, they provide neither enough expressiveness to specify them nor enough accuracy to allow their subsequent automation.

Finally, note that, unlike the above works, our approach makes a step further for providing an integrated approach, covering all stages of the development process from requirement elicitation to service execution. We concretely exploit knowledge gathered by UCD techniques to derive the executable models that represent the AAL services.

III. Methodology to Support AAL

To achieve the automation of AAL services, we propose a model-driven development methodology that supports their development from requirements elicitation to their execution. The methodology is proposed considering the following main aspects:

- AAL services must be automated according to specific medical guidelines and to the requirements provided by medical professionals and patient responsible persons. This is essential so that the tasks that are automated can assist to the patients and medical staff (end-users of the system) in the best way. If this information is not taken into account, the AAL services could be very intrusive for the end-users of the system, bothering them, interfering in their goals, or even being dangerous. For instance, according to medical guidelines, a softer room illumination and relaxing music playing may help to make patients more relaxed. If these guidelines are not considered, medical or security staff would be needed every time an aggressive behaviour is detected. Due to the medical context, and the imprecise and ambiguous nature of patient behaviour, it is very difficult for a system to sense or infer this information [20]. Therefore, **the participation of the corresponding end-users is** absolutely **necessary** for supporting AAL.
- **The AAL services must be context-adaptive**. Context information is essential to be able to execute the assiting tasks in the opportune situation. Therefore, AAL services must be described in a context-adaptive way. For instance, if a health anomaly is detected in a patient when s/he has fallen, the appropriate nurses or doctors should be inmediately notified; however, if the patient health is not in risk, it is enough to notify the nearest available caregivers.
- **The evolution of the AAL services must be facilitated after system deployment**. Some AAL services might never change; however, most of them will. Patients' behaviour and health may change over time and the automated services to support them need to be adapted to these changes. Otherwise, the system may become useless and intrusive. Since these types of changes cannot be anticipated at design time, the automation of AAL services must be performed in such a way that their evolution after system deployment is facilitated at runtime.

In order to deal with these aspects, the methodology we propose consists on the following steps (see Figure 1):

- *Requirements Elicitation: AAL Service Identification.* In order to capture of medical guidelines and to properly capture the end-users requirements, we use UCD tecniques to facilite the participation of the specific end-users. We use UCD techniques because they give extensive attention to needs, wants, and limitations of users at each stage of the development process, which is crucial in the design of AAL systems. This methodology step will be further explained in Section IV.
- *AAL Service Modelling.* This activity consists of modelling the AAL services that must be automated by the system. An AAL service is a set of tasks that are habitually performed in a certain context for assisting patients and medical staff. The following steps must be followed to specify the identified AAL services:
  - Context modelling. Analysts specify the context properties on which the AAL services depend, create the necessary rules to infer properties values, and set the property values that need to be manually introduced.
  - AAL service modelling. Using the context-adaptive task model, the analysts specify the AAL services to be automated according to the context previously specified. Each AAL service consitutes a coordination of tasks and is specified as a task hierarchy in which the service represents the highest task in the hierarchy. Each service needs to be progressively broken down into simpler tasks until they can be automatically executed by a pervasive device.
  - Modelling validation. The AAL services' modelling is validated with the end-users to ensure that the tasks will be automated according to patients' needs and medical guidelines. Thus, following an iterative process, the service modelling (and if needed the context modelling), must be refined with the end-users participation until they agree with the specified AAL services. It is important to note that, in this way, the modelling is complemented by both the knowledge of the system analysts, which contributes to improving the performance of the identified AAL services; and the knowledge of the medical professionals and persons responsible of the patients', which contributes to taking into account the needs and demands of the end-users. After validating the service modelling with the end-users, analysts also validate that the models are correctly formed and without inconsistencies.
  - Device linkage: once the modelling has been validated, the analysts link each leaf task with a perva-

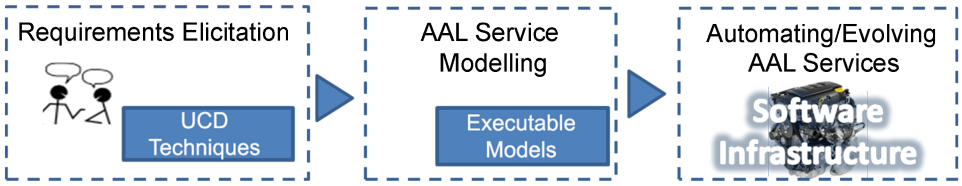


Fig. 1. Methodology for supporting AAL Services

sive device that can carry it out. Pervasive devices can control the objects in the environment (e.g., switching lights on, activating the security alarm, etc.) and sense context information (e.g., detection of presence, measurement of temperature, etc.). Specifically, we consider a pervasive device to be an entity that provides a coherent set of functionality which is described in terms of atomic operations (or methods). Thus, the linkage between leaf tasks and the corresponding pervasive device is made in the task model by indicating the name of the corresponding pervasive device and its operation. The implementation of these devices is out of the scope of this paper; an approach such as PervML [21][22] could be used for developing them.

Since the used context-adaptive task models are executable models [23], this step finishes the AAL service development. This also allows that the specified AAL services can be validated by using prototypes. This would require that the automation of the specified AAL services is done in a simulation mode. This mode should allow analysts to cause context changes and to easily observe the execution of the services according to context.
This methodology step will be further explained in Section III.

- *AAL Service Automation*. To enable the automation of the specified AAL services in the opportune context, the following two steps must be performed:
  - Deployment of the system in the target platform. To deploy the system, analysts install each component of the software infrastructure in an OSGi server. We use an OSGi server [24] because it provides numerous benefits and facilities to make dynamic updates, to easily reuse components, or to deploy the system. In addition, the context model and the task model where the AAL services are specified must be saved in the folder where OSGi is installed.
  - Running the system. To run the system, analysts start the components installed in OSGi. From this moment, the context is continuously monitored and the specified AAL services are automated in the appropriate context. It is important to note that, since the models are directly interpreted, they are the only representation of the AAL services to be automated. This facilitates their understanding, maintenance and evolution.

  This methodology step will be further explained in Section VI.
- *Evolution of the AAL services if needed.* Medical staff and patient behaviour as well as patients' health may change over time and the AAL services that are automated may become obsolete or useless. If this happens, the automated AAL services can be evolved according to the new requirements. To allow this evolution, evolution mechanisms are provided. This methodology step will be further explained in Section VI.

In the next sections we explain how these steps are supported.

IV. AAL Service Requirement' Elicitation

In order to capture the requirements for automated services, we use UCD techniques. AAL services can be defined for specific patients that suffer from very concrete disabilities or for a group of patients that share a same profile. Therefore, the elicitation process is based on the description of personas and technological scenarios. Their conjunct use increases the ability to identify problems and exceptional cases [25] and to envisage the system to be [26].

Personas are descriptive models of system-to-be users based on behavioural data, derived from patterns observed during interviews, with the aim of representing the diversity of observed motivations, behaviours, and mental models [27]. Examples can be found in Figure 2.

In the context of AAL, a Persona may represent a single patient with specific needs, a group of patients that share same needs, or a medical profile (a doctor, a nurse, a caregiver, etc.). In the case of being either a group of patients or a medical profile, they are personified through a fictitious character that represent them. For the former, the character has the needs that are shared by all of them; for the latter, the character represents the professional skills that are shared by all the people of this profile. This facilitates a shared understanding of who the patients and medical professionals are, and what they need or can provide in order to make decisions about AAL services. In addition, Personas provide a powerful tool for communicating between computer analyst and medical professionals in order to develop and evaluate AAL services. Figure 2 shows two examples of Persona. The first one, Maria, represent a patient with senile dementia; the second one, Sabrina, represent an experienced caregiver.

Personas are complemented with *technological scenarios* that illustrate how they interact with the system. Technological

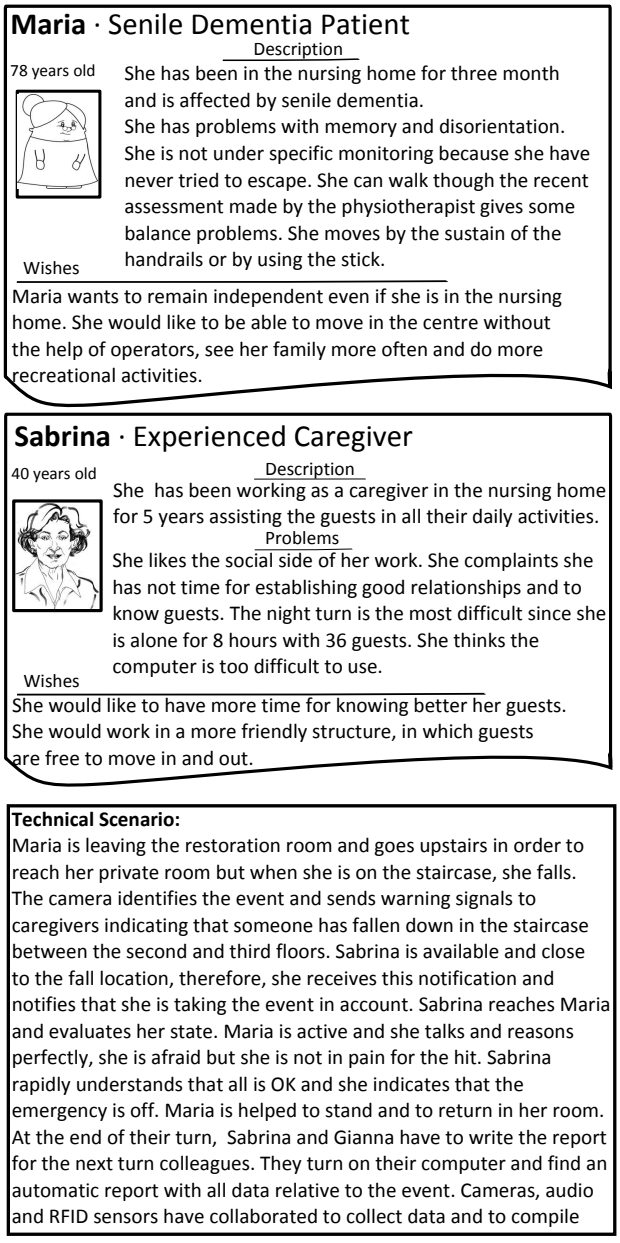


Fig. 2. Example of personas en technical scenario

Scenarios are short narrative stories that represent Personas in their context, supported by the envisaged technology (see Figure 2). Many techniques can be profitable for envisioning system functionalities and services; examples are: internal meetings, brainstorming, focus groups and scenarios. The output of these techniques are the technological scenarios that concretely describe the behaviour of services as experienced by specific, though fictional, users. Stories help the design teams in negotiating a shared representation of the domain and hence a more effective collaborative elicitation of requirements. In this work, scenarios describe how the system interact with patients and medical professionals when specific situations occur. For instance, the scenario presented in the bottom side of Figure 2 explain how the system must act when a patient fall is detected.

V. AAL Service Modelling

Once personas and scenarios are defined, the AAL services to be automated must be identified and modelled from the obtained requirements. According to the complexity of the system, the modelling can be done manually by directly analysing the requirements, or it could be done by using some type of transformation technique such as the one presented in [28], which is based on the definition of intermediate goals.

The modelling of the AAL services is performed by specifying two models: a context model (which specifies the context on which the services to be automated depend), and a context-adaptive task model (which describes the tasks that must be carried out for each service according to the context described in the context model).

The context model represents the context relevant for the AAL services so they can be executed in a non-intrusive way. Specifically, the model represents information regarding Patients, Medical Staff, Locations, Environment Properties, Policies, Temporal Properties, Services and Events. The model is specified in the Ontology Web Language (OWL)[29], which is an ontology markup language W3C standard that greatly facilitates knowledge automated reasoning and inference. Thus, the classes of the ontology are defined as OWL classes, their relationships as OWL object properties, their attritutes as datatype properties, and the context specific to the system is defined as OWL individuals. For specifying the context model, we use Protégé [30], which is a free open source ontology editor. A context model example created using Protege is shown in Figure 3. From left to right, this model shows some context classes (such as Patient, Caregiver, Location, EnvironmentProperty. etc.), some relationships among these classes (such as locatedIn and isRelatedWith), some data properties of the classes (such as email and id), and some individuals (such as RestorationRoom and RestorationRoom_NoiseLevel). This model contains the following types of context information:

- Manually introduced, such as the personal information of the patients and caregivers. This information is introduced by the analysts.
- Automatically captured, such as locations and health parameters of patients. This information is captured by the context manager that will be explained in the next section.
- Automatically inferred, such as healthAnomaly, which is set to true when a health parameter is outside the normal values. This information is automatically set by inference rules created in the ontology.

Using the context model, the task model describes context-adaptive AAL services precisely and at a high level of abstraction. As an example, Figure 4 shows the modelling of the AAL service that supports the scenario of patient falls. The root task of the hierarchy represents the service and is associated to a context situation, which indicates the context conditions

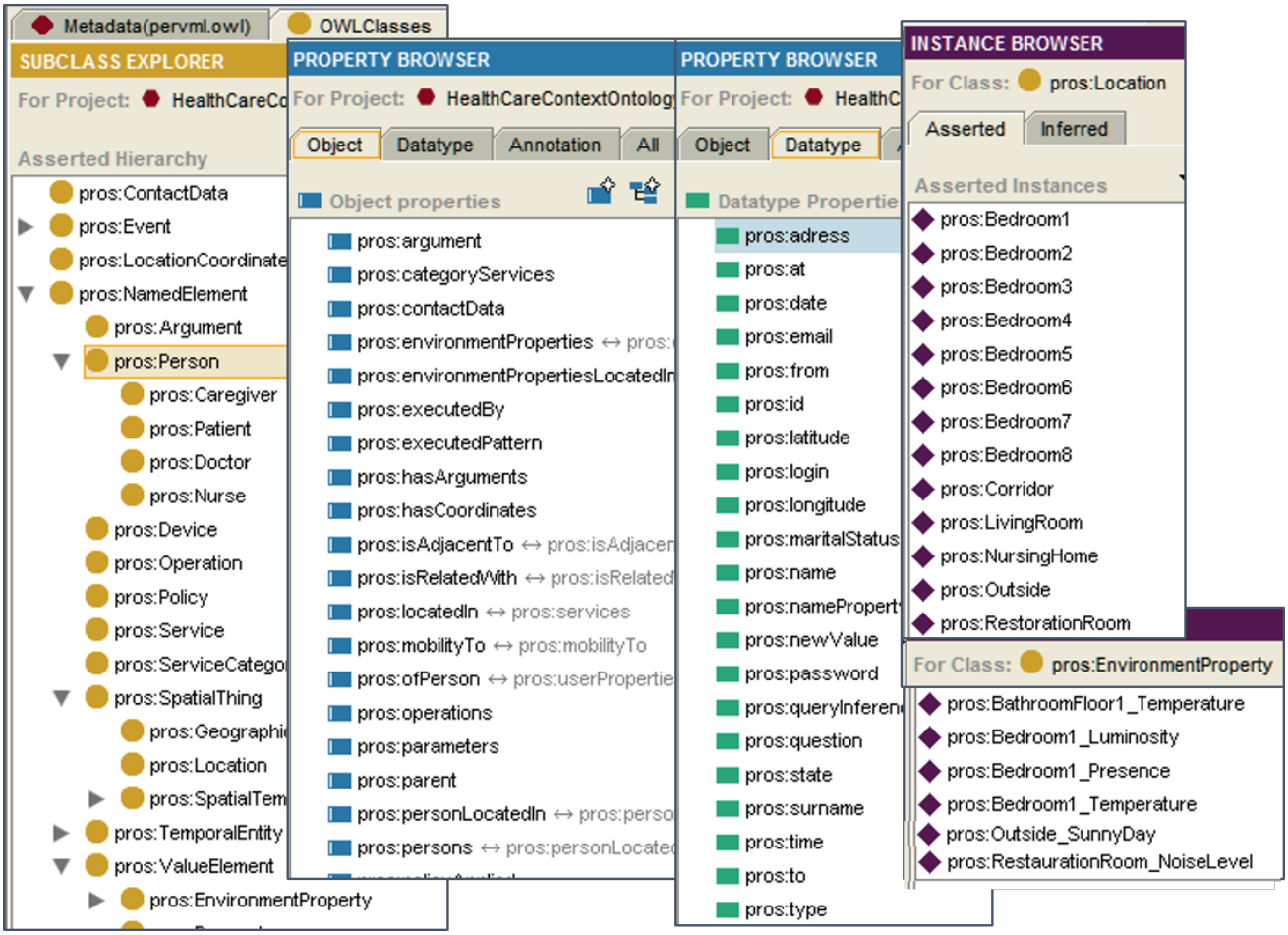


Fig. 3. Context Model Example

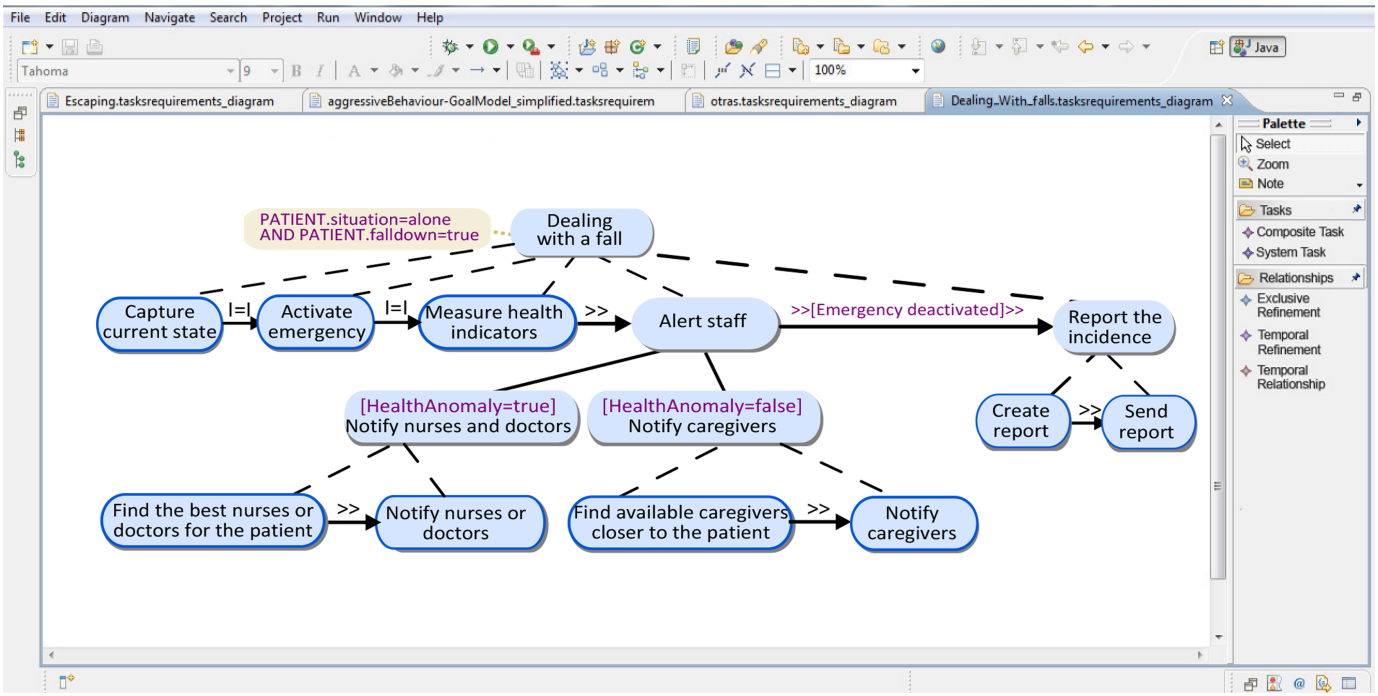


Fig. 4. Service modelling using the context-adaptive task model: dealing with a fall AAL service

whose fulfilment starts the execution of the service (*PATIENT.situation=alone AND PATIENT.falldown=true*). The root task is broken down into simpler tasks (*Capture current state, Active emergency, Measure health indicator, Alert staff, Report the incidence*). An intermediate task must be broken down until the leaf tasks can be executed by an available pervasive device. Each leaf task must be related to a pervasive device that can carry out the task. For instance, the active emergency task is associated to a pervasive device that interacts with the emergency system to turn it on. This relation is established by simply indicating the device identifier.

If the tasks of the same parent are related to each other, they are carried out in a sequential order according to the indicated temporal relationships. These relationships may depend on context. Thus, in the example, the current state is captured at the same time (which is indicated by the temporal relationship $|=|$) that the emergency is activated and the health indicator are measured; after that (temporal relationship $>>$) the staff is alerted; and finally, the incident is reported when the emergency is deactivated (temporal relationship with a context condition indicated between brackets $>>[]>>$).

In addition, a task can have a context precondition (represented between brackets before its name), which defines the context conditions that must be fulfilled so that the task is performed (e.g., the notify nurses and doctors task is only executed if HealthAnomaly is true). If the tasks of the same parent are not related to each other, only the first task whose context precondition is satisfied is executed. For instance, if the Notify nurses and doctors is executed its sibling tasks Notify caregivers is not.

For user-intensive systems, ontology classes can be used in the context conditions instead of individuals. The condition is satisfied when it is satisfied for one of the individuals of the used class. For instance, the *Dealing with a fall* AAL service has to be executed for every patient. As shown in Figure 4, instead of specifying the same AAL service for each patient, we specified the service once and used the situation and falldown context properties of the PATIENT class in its context situation, indicating by using capital letters that it is an ontology class and not an individual. Thus, the context condition has to be checked for every individual of the Patient class and is activated if any patient fulfils the condition.

For facilitating the specification of the task model, we developed a graphical editor using the Eclipse platform, and the EMF and GMF plugins. By using this editor, the model can be graphically edited as shown in Figure 4. These descriptions are stored in XMI (XML Metadata Interchange), which is machine-interpretable at runtime.

More details about the context model, and the task model and its editor can be found in [23].

VI. AAL Service Automation and Evolution

In order to automate the AAL services as specified in the models and facilitate service evolution after system deployment, we have used the software infrastructure presented in [1]. This infrastructure, which is shown in Figure 5, directly interprets the context model and the task model at runtime to automate the AAL services as described. This infrastructure is built on top of the pervasive devices used to sense context changes and to perform the leaf tasks of the AAL services specified in the models.

The software infrastructure is composed by the following components (see Figure 5): mechanisms for managing the models at runtime, a context manager, and an automation engine.

Mechanisms for managing the models at runtime. In order to manage the context model and the task model at runtime, we have designed and implemented Ontology-based Context model management mechanisms (OCean) and Model-based User Task management mechanisms (MUTate). These mechanisms can be downloaded from http://www.pros.upv.es/art/.

- OCean: The context on which the behaviour patterns depend is specified in the context model as OWL individuals. Thus, in order to manage these individuals, a set of Ontology-based Context model management mechanisms (OCean) is needed. OCean allows, for instance, updating the individuals of the context model, creating a new individual (e.g., the *idealTemperature* individual of the *Preference* ontology class), and reading its properties or modifying them when needed. We have extended OCean to support the checking of context conditions in user-intensive services. In this way, when an ontology class (represented in capital letters) is used in a condition, the condition is considered as satisfied when it is satisfied for one of the individuals of the class.
- MUTate: In order to support the management of the task model, a set of Model-Based User Task management mechanisms (MUTate) is needed. MUTate allows, for instance, searching for an AAL service that have to be executed, obtaining its related context situation, adding new tasks to an AAL service, and creating an AAL service.

OCean and MUTate provide access to the models by using the same vocabulary defined in the context ontology and the task model, respectively. It is important to note that, in this way, they provide high-level abstraction mechanisms that facilitate the interaction with the models without the need to stop the system. Both, OCean and MUTate are needed in order to achieve the automation and evolution of the specified AAL services.

The **context manager** monitors the pervasive sensors install in the environment to detect and process context changes. The context manager also updates the context model according to the detected context changes. The context manager uses OCean to perform this update at runtime.

The **automation engine**, named MAtE (Model-based Automation Engine), is in charge of automating the AAL services in the opportune context by interpreting the models at runtime using MUTate. To automate an AAL service, MAtE executes its system tasks by taking into account the current context,

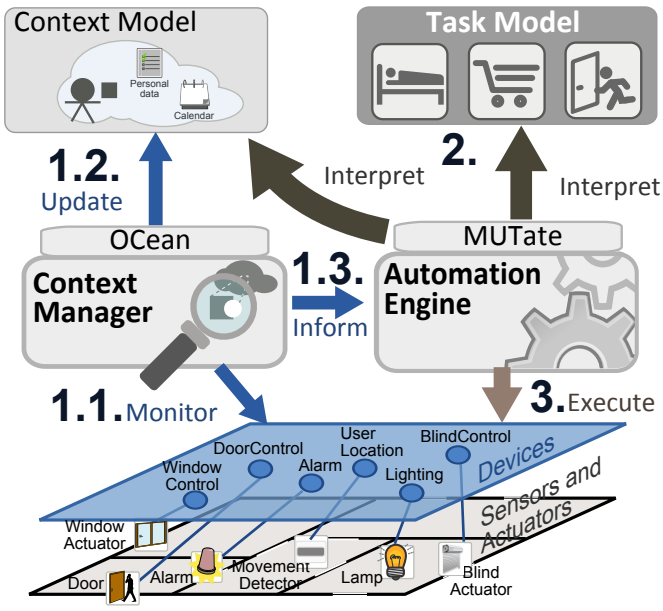


Fig. 5. Runtime infrastructure

their relationships and their refinements. Each system task is executed by MAtE using the pervasive service related to it.

A. The Software Infrastructure in Execution

The software infrastructure executes the AAL services as described in the task model by performing the following steps (see Figure 5):

1) *Detect context changes:* A context change is physically detected by a sensor. The context manager monitors all the sensors to check context changes (step 1.1 in Figure 5). For instance, the context manager monitors periodically the location of patients and whether or not a fall is detected. When a change is detected (e.g., a patient has fallen down) the context manager updates the context model using OCean (step 1.2 in Figure 5) and notifies the automation engine about the context change (step 1.3 in Figure 5).
2) *Interpret context situations:* After receiving the notification of a context change, the engine analyzes the context situations of the AAL services specified in the task model to check if any of them depend on the context change. Then, by making use of the context manager, the engine checks if any of those context situations are fulfilled. For instance, when the context manager notifies the engine that a user has fallen down, the engine gets the context situation that depends on this aspect, such as the one shown in Figure 4. If the fallen user is also alone, then the context situation of this AAL service is satisfied.
3) *Execute the AAL services:* The engine executes the AAL services whose context situation is satisfied. The engine uses the context manager to check the context conditions. To execute each AAL service, the engine executes its leaf tasks according to their refinements, their context conditions in the current context, and their temporal relationships. For instance, when the context situation of the AAL service focused on detecting falls (see Figure 4) is satisfied, the engine captures the current state, activates the emergency, and measures the health indicators by using the corresponding pervasive devices. Next, the rest of tasks are executed according to the task plan defined in the task model (see Figure 4).

If the requirements of the AAL services change over time, OCean and MUTate can be used to modify the models at runtime to perform the required evolutions. Using these mechanims, the services can be evolved by using concepts of high-level of abstraction such as user tasks, patient, or context situations. For instance, in order to make sure that an automated service executes one task instead of another, analysts just need to replace the corresponding task; in order to change the order in which tasks must be executed, analysts just need to modify temporal relationships; in order to change the situations in which services must be executed, analysts just need to change a context condition.

In addition, models are decoupled from the implementation of pervasive devices since the models just use identifiers to reference these services. This also facilitates the evolution of the assisting services since they are independent of pervasive technologies and internal implementation aspects.

B. Implementation Details

The *context manager* and the *automation engine* are implemented in Java/OSGi technology and are run in an OSGi server together with the pervasive devices.

Using OSGi, the context manager can listen to the changes produced in the services to detect context changes and can also inform the engine when a change is detected. To execute a task, the engine searches for the pervasive device associated to the task in the OSGi server by using its service registry. Then, the engine executes the corresponding device method by using the Java Reflection capabilities.

To manage the task model at runtime, MUTate uses the EMF Model Query plugin that allows a system to work with any model by querying its structure at runtime. To manage the context repository at runtime, OCean uses the OWL API 2.1.1, which provides facilities for creating, examining, and modifying an OWL model; and the Pellet reasoner 1.5.2., which allows the OWL model to be queried. More technical details can be found in [31].

VII. Validation of the Proposal

Following the guidelines provided in [32], we have developed a case-study based evaluation where the automations needed for the ACube research project were created. ACube is a project founded by the local government of the Autonomous Province of Trento in Italy. ACube aims at designing an automated user intensive system to be deployed in nursing homes as a support to medical and assistance staff.

Using the proposed methodology, we designed and developed an automated system for a specific nursing home subject of study in the ACube project. The main goal of the system was to automate AAL services that were usually performed by medical and assistance staff is to help them to make their work more efficiently in order to enhance their quality of work and the quality of life of their patients. By automating AAL services, the tasks of medical and assistance staff can be greatly reduced freeing them so that they can spend more time with their patients. In addition, the tasks that medical and assistance staff perform can be improved to be more efficient because the tasks can be previously analysed and also can be carry out even when none caregiver is present.

According to the methodology requirements (see Section III), we evaluated the following research questions:

1) Does the approach facilitate end-users participation in the AAL service design to take into account medical professional guidelines and the specific requirements from medical professionals and patients?
2) Does the approach correctly automate the specified AAL services in a context-adaptive way?
3) Does the approach allow the automated AAL services to be evolved after system deployment?

We now summarize the results of this evaluation. More details can be found in [31].

A. Evaluating End-Users' Participation

Our approach makes use of design models at runtime. It provides a task model that describes the AAL services by using concepts of a high-level of abstraction that are close to the domain and to end-users' knowledge (concepts such as task, preference, location, patient, etc.). This helps end-users to participate in the service description since it allows them to focus on the main concepts (the abstractions) without being confused by low-level details [33].

The ACube consortium had a multidisciplinary nature, involving software engineers, sociologists and analysts, and it is characterized by the presence of professionals representing end-users directly engaged in design activities. For developing the case study, we interact with some of these professionals which were responsible of analysing the requirements for the AAL services.

After describing the personas and their scenarios, we analysed them and designed the AAL services that the system should automate for supporting each one of the scenarios.

We then discussed the designed task models with the professionals in charged to include medical guidelines and validate the tasks with them. To deal with these discussions, we briefly explained the main concepts of the task model and the behaviour of two AAL services. Then, we checked the model comprehension using a short oral questionnaire that asked questions such as: how many tasks will be executed in this service?; when will this service be activated?; which is the next task that will be executed?. These questions make the users reason about the model, which is a recommended technique to evaluate the understanding of a model [34]. We found that the task model is very useful in discussing and validating the AAL services to be automated since it was very intuitive for them after explaining a couple of examples. If something was not specified the way the professionals considered suitable, we refined the model to fulfil their requirements. We repeated this process until the professionals agreed with the specification. This allowed us to describe the AAL services by taking into account the medical guidelines and requirements provided by the professionals.

Figure 4 shows the final specification of the AAL service to support patient falls. These tasks can be described as follows: the service is activated when it is detected that a patient falls and none of the caregivers or medical staff is around. When this happens, the system captures the current context state, activates the emergency state and measures the health of the patient. Then, the system alerts either the medical staff if the patient health is critical or the nearest caregivers if the patient is fine. Finally, when the emergency is under control, a report about the incidence is created and sent to the involved staff so that they can validate it.

B. Evaluating AAL Service Automation in a Context Adaptative way

To execute the described AAL services **in a context-adaptive way**, the task model describes each AAL service as a coordination of tasks that are performed in the opportune context, i.e., in a context-adaptive way. In addition, in order to be aware of the current context and to be able to automate the AAL services accordingly, the software infrastructure provides a context manager. It dynamically manages the context changes produced at runtime by using the context repository.

To validate the context adaptation, we put the system into operation to automate the described AAL services after the task models were validated with the end-users. We used a device simulator and an Equinox distribution (which is the OSGi implementation of Eclipse) running in the PC. To support the functionality needed to execute the described AAL services, we developed the required simulated pervasive devices (a total of 17 different pervasive devices). See [31] for more details about these devices.

We then evaluated the feasibility of our software infrastructure. Using the running system, we passed the JUnit tests developed to check that the specified AAL services were correctly automated as specified in the models. Since the automation of the AAL services are triggered as a response to context changes, we caused these context changes by changing the state of the sensors using the simulator. We changed the state of the sensors simulating the scenarios of the requirement elicitation phase. For instance, to enable the *Dealing with a Fall* AAL service, we simulate that a patient fell down when she was alone. This makes the context situation of the AAL service fulfil (see Figure 4).

In the same way, we simulated the rest of the scenarios of the case study and executed the prepared JUnit tests. For all of them, we checked that they were executed as specified in the models.

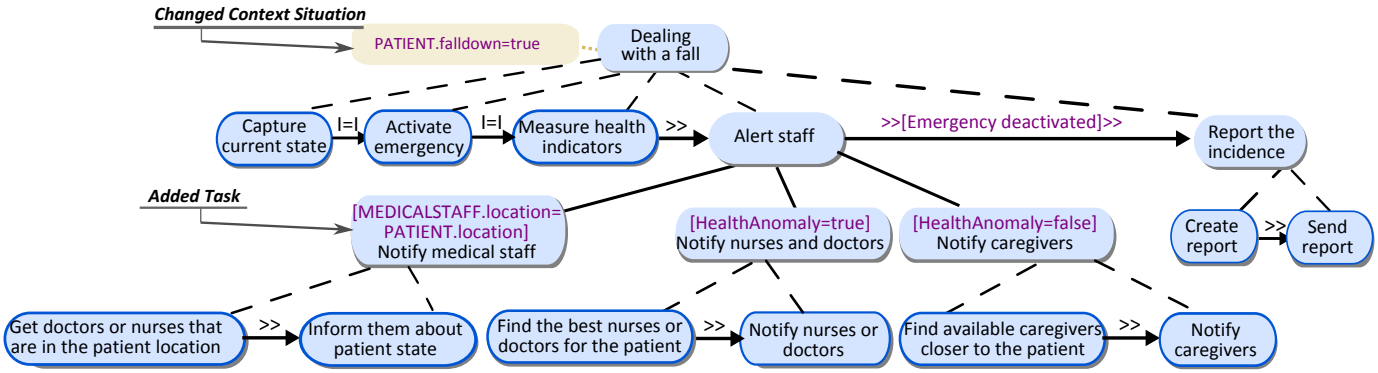


Fig. 6. Example of a AAL services' evolution

Furthermore, we evaluated the system performance. Models are manipulated at runtime by OCean and MUTate. Therefore, these operations have to be efficient enough so that the system response is not drastically affected. In order to measure the system response, we quantified the temporal cost of the operation done with randomly generated large models. We used a laptop intel core i7-4600U, 2.70GHz and 8GB of RAM, with Windows 8.1 end Eclipse Modelling Kepler 32 bits. We used the context model presented in Section V and an empty task model to be randomly populated by means of an iterative process. The context model was populated with 100 new context individuals in each iteration, while the task model was populated with one new AAL service whose task structure formed a binary tree, varying the depth and width of the first level of the tree each iteration.

For each iteration, we tested all the model operations 20 times and calculated the average temporal cost of each one. As an example, the operation of OCean with the highest temporal cost was the operation to get a specific context individual, which took less than 0.2 milliseconds for 6000 individuals. The temporal cost of the MUTate model operations with the highest cost are the operations for getting, updating, and deleting a task. These costs are very similar since all of them run the same query to obtain the corresponding task. Even with a model population of 45612 tasks, these model operations provided a fast response (less than 8 milliseconds). Therefore, the results show that the response time is not drastically affected when models with a high number of instances are used.

C. Evaluating AAL Service Evolution after System Deployment

To automate the described AAL services in such a way that **their evolution after system deployment is facilitated**, the automation engine directly interprets the task model at runtime. The model is machine-processable and precise enough to be executed. Therefore, when a context change is detected by the context manager, it informs the engine. The engine then reads the AAL service information from the task model and executes the corresponding pervasive services according to context. With this strategy, the task model is the only representation of the AAL services to be automated. This allows them to be adapted by simply updating the model. As soon as it is changed to evolve the AAL services, the changes are also taken into account by the engine.

To validate this runtime evolution, we changed the task model using OCean and MUTate to perform the following types of updates: add, delete and modify tasks; modify context situations, task order, context preconditions, temporal relationships, etc.

After each update, we simulated the fulfilment of the context situations of the AAL services and applied the JUnit tests again to check that the tasks were correctly executed according to the performed evolution. For instance, Figure 6 shows an example of these evolutions. It shows how the *Dealing with a Fall* AAL service has been modified to be executed regardless if the patient is alone or not. In addition, it has been added the Notify medical staff task that is executed when there is medical staff in the same location of the patient; if so, the doctors or nurses are notified about the patient state since they can look after the patient straightaway. If there is not medical staff in the same location, then the system follows the same plan that was specified in the previous version (if any health anomaly is detected, then the appropriate doctors or nurses are notified; otherwise, the closer available caregivers are notified). For each performed evolution, we applied again the JUnit tests checking that all the AAL services were correctly executed.

VIII. Conclusions and Further Work

In this work, we have presented and evaluated a model-driven approach that achieves the automation of services for improving AAL. These AAL services are represented in high-level abstraction context-adaptive task models that are executable, i.e., they are directly executed by a software infrastructure that automates the AAL services as specified in the models. This considerably facilitates the further evolution of the AAL services by directly changing the models (i.e., at the modelling level) at runtime, which is one of the top challenges in software evolution research [35]. As soon as the models are changed to evolve the AAL services, the changes are also taken into account by the automation engine.

As further work, we plan to develop an end-user tool that allows medical professionals to create and evolve AAL services by their own. This tool will provide medical professionals with intuitive user interfaces that let them describe assisting services by using their own knowledge and concepts. Then, a model-to-model transformation will be applied in order to generate context-adaptive task models. As a previous experience in the development of this type of tools we developed an end-user tool [36] focused on the adaptation of smart home systems. This tool allows home inhabitants to create and evolve the routine tasks that they want to have automated. Our goal is to reuse all this experience to create a similar tool focused on AAL environments.

ACKNOWLEDGMENT

This work has funded by the Research Fund KU Leuven.

REFERENCES

[1] E. Serral, P. Valderas, and V. Pelechano, "A software infrastructure for executing adaptive daily routines in smart automation environments," in *ADAPTIVE 2013, The Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications*, 2013, pp. 30–35.

[2] H. Steg, H. Strese *et al.*, "Ambient assisted living–european overview report," 2005.

[3] B. Takács and D. Hanák, "A mobile system for assisted living with ambient facial interfaces," *International Journal on Computer Science and Information System*, vol. 2, pp. 33–50, 2007.

[4] A. Association. (2014) Ambient assisted living (aal) joint programme. [Online]. Available: http://www.aal-europe.eu/

[5] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman, "Creating an ambient-intelligence environment using embedded agents," *IEEE Intelligent Systems*, vol. 19, no. 6, pp. 12–20, 2004.

[6] P. Rashidi and D. J. Cook, "Keeping the intelligent environment resident in the loop," in *IE 08*, 2008, pp. 1–9.

[7] K. Henricksen, J. Indulska, and A. Rakotonirainy, "Using context and preferences to implement self-adapting pervasive computing applications," *Software: Practice and Experience*, vol. 36, no. 11-12, pp. 1307–1330, 2006.

[8] M. García-Herranz, P. Haya, and X. Alamán, "Towards a ubiquitous end-user programming system for smart spaces," *Journal of Universal Computer Science*, vol. 16, no. 12, pp. 1633–1649, 2010.

[9] A. Cooper, R. Reimann, and D. Cronin, *About face 3: the essentials of interaction design*. John Wiley & Sons, 2012.

[10] A. Dohr, R. Modre-Opsrian, M. Drobics, D. Hayn, and G. Schreier, "The internet of things for ambient assisted living," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*. Ieee, 2010, pp. 804–809.

[11] F. G. Miskelly, "Assistive technology in elderly care," *Age and ageing*, vol. 30, no. 6, pp. 455–458, 2001.

[12] P. P. Portal. (2014) Persona project portal. [Online]. Available: http://www.aal-persona.org

[13] J. A. Kientz, S. N. Patel, B. Jones, E. Price, E. D. Mynatt, and G. D. Abowd, "The georgia tech aware home," in *CHI'08 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2008, pp. 3675–3680.

[14] U. of Illinois at Urbana-Champaign. (2014) I-living: Assisted living project. [Online]. Available: http://lion.cs.uiuc.edu/assistedliving

[15] J. Nehmer, M. Becker, A. Karshmer, and R. Lamm, "Living assistance systems: an ambient intelligence approach," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 43–50.

[16] E. Serral, P. Valderas, and V. Pelechano, "Improving the cold-start problem in user task automation by using models at runtime," in *Information Systems Development*. Springer, 2011, pp. 671–683.

[17] D. Cook and S. Das, *Smart environments: Technology, protocols and applications*. Wiley-Interscience, 2004, vol. 43.

[18] C. Pribeanu, Q. Limbourg, and J. Vanderdonckt, "Task modelling for context-sensitive user interfaces," *Interactive Systems: Design, Specification, and Verification*, pp. 49–68, 2001.

[19] R. Huang, Q. Cao, J. Zhou, D. Sun, and Q. Su, "Context-aware active task discovery for pervasive computing," in *International Conference on Computer Science and Software Engineering*, 2008, pp. 463–466.

[20] F. M. Reyes, "Issues of sensor-based information systems to support parenting in pervasive settings: A case study," *Emerging Pervasive and Ubiquitous Aspects of Information Systems: Cross-Disciplinary Advancements*, p. 261, 2011.

[21] J. Muñoz, E. Serral, C. Cetina, and V. Pelechano, "Applying a model-driven method to the development of a pervasive meeting room," in *ERCIM News*, April 2006, pp. 44–45.

[22] E. Serral, P. Valderas, and V. Pelechano, "Towards the model driven development of context-aware pervasive systems," *Special Issue on Context Modelling, Reasoning and Management of the Pervasive and Mobile Computing (PMC) Journal*, 2010.

[23] E. Serral, P. Valderas, and V.Pelechano, "Context-adaptive coordination of pervasive services by interpreting models during runtime," *The Computer Journal*, vol. 56, no. 1, pp. 87–114, 2013.

[24] (2014) Osgi. http://www.osgi.org/.

[25] A. Sutcliffe, N. Maiden, S. Minocha, and D. Manuel, "Supporting scenario-based requirements engineering," *IEEE Trans. on Soft. Eng.*, pp. 1072–1088, 1998.

[26] C. Rolland and C. Salinesi, "Supporting Requirements Elicitation through Goal/Scenario Coupling," in *Conceptual Modeling: Foundations and Applications*. Springer, 2009, p. 416.

[27] A. Cooper, R. Reimann, and D. Cronin, *About face 3: the essentials of interaction design*. Wiley India Pvt. Ltd., 2007.

[28] E. Serral, L. Sabatucci, C. Leonardi, P. Valderas, A. Susi, M. Zancanaro, and V. Pelechano, "Incorporating users into ami system design: From requirements toward automation," in *Information Systems Development*, R. Pooley, J. Coady, C. Schneider, H. Linger, C. Barry, and M. Lang, Eds. Springer New York, 2013, pp. 499–511.

[29] Smith, Welty, and McGuinness, "Owl web ontology language guide," 2004.

[30] N. F. Noy, M. Crubézy, R. W. Fergerson, H. Knublauch, S. W. Tu, J. Vendetti, M. A. Musen *et al.*, "Protege-2000: an open-source ontology-development and knowledge-acquisition environment," in *AMIA Annu Symp Proc*, vol. 953, 2003, p. 953.

[31] E. Serral, "Automating routine tasks in smart environments. a context-aware model-driven approach," Ph.D. dissertation, Technical University of Valencia, DSIC, 2011.

[32] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[33] F. Paternò, "From model-based to natural development," *HCI International*, pp. 592–596, 2003.

[34] A. Gemino and Y. Wand, "Evaluating modeling techniques based on models of learning," *Communications of the ACM*, vol. 46, no. 10, October 2003, modeling comprehensibility.

[35] T. Mens, "The ercim working group on software evolution: the past and the future," in *IWPSE-Evol workshops*. ACM, 2009, pp. 1–4.

[36] E. Serral, F. Pérez, P. Valderas, and V. Pelechano, "An end-user tool for adapting home automation to user behaviour at runtime," *UCAmI'10*, pp. 201–210, 2010.

Testing Self-Adaptive Software: Requirement Analysis and Solution Scheme

Georg Püschel, Sebastian Götz, Claas Wilke, Christian Piechnick, and Uwe Aßmann
Software Technology Group, Technische Universität Dresden
Email: {georg.pueschel, sebastian.goetz1, claas.wilke, christian.piechnick, uwe.assmann}@tu-dresden.de

***Abstract*—Self-adaptive software reconfigures automatically at run-time in order to react to environmental changes and fulfill its specified goals. Thereby, the system runs in a feedback loop which includes monitoring, analysis, adaptation planning, and execution. To assure functional correctness and non-functional adequacy, verification and validation is required. Hence, the feedback loop's tasks have to be examined as well as the adapted system behavior that spans a much more complex decision space than traditional software. To reduce the complexity for testers, models can be employed and later be used to generate test cases automatically—an approach called Model-based Testing. Alternatively, the models can be executed directly for which simulation-based validation can be employed. For both methods, an engineer has to specify validation models expressing the system's externally perceivable behavior as well as expectations derived from requirements. In this paper, we perform a Failure Mode and Effects Analysis on a generic perspective on self-adaptive software in order to derive additional requirements to be coped within test modeling. Besides functional requirements, we discuss non-functional requirements in particular. From these requirements, a reference solution scheme is derived that can be used to construct and evaluate validation methods for self-adaptive software. For illustration, we provide an example from the home robotics domain.**

***Keywords*—Self-adaptive Software; Model-based Testing; Simulation; Failure Modes and Effects Analysis.**

I. Introduction

In our original work [1], we studied requirements that have to be coped with in testing self-adaptive software (SAS, [2]). This certain kind of system reconfigures automatically at run-time according to sensed context changes. Thus, it is able to effectively and efficiently fulfill its specified goals under changing conditions. For instance, a beneficial application area of SAS are Cyber-physical Systems (CPS, [3]). CPS reflect physical objects, software, and their interplay in order to reason about them. Due to the SAS's ability to automatically adapt to changes in such a context, a CPS may operate autonomously without the requirement of a strictly controlled factory environment. Thus, the development of sophisticated methods and technologies for developing SAS may help to make systems like autonomous cars and home robotic systems become reality.

The user of an SAS can delegate tasks to the system at run-time. Such tasks do not have to be known to the system in advance. Several systems support descriptive formats like goal models or rules for this purpose. Furthermore, there may be unanticipated events in the environment that have to be considered in the SAS's decision process as well as external adaptation mechanisms that change the system structure in an unforeseen manner. In consequence, an SAS engineer has to be aware of several unpredictable behaviors that may impact design decisions.

However, the manufacturer of a system has to give promises to customers about its correctness (e.g., in form of a certificate). Therefore, at least a subset of the SAS capabilities have to be verified or validated before delivery. As each step of the development life cycle is equipped with a limited budget, it is difficult or even impossible to examine the system's correctness completely. Thus, a more scenario-based examination of the SAS is preferable. Additionally, self-testing [4] or even self-verification [5] mechanisms can be built in the system and triggered at the point in time when the system is adapted.

Another problem is that SAS engineers face additional complexity. During each test scenarios' workflow, the test steps can be adapted to changing context situations. In consequence, in the worst case, the state space of the system is combinatorial between adaptation state and workflow state [6, p. 17]. Furthermore, it has to be considered that a context change, that causes an adaptation, has to be taken into account in order to reproduce specific adaptation states. Due to all these complexity factors, the difficulty of applying verification methods like model checking increases enormously. Hence, in our work, we instead focus on determining validation techniques that provide appropriate abstraction means for SAS' state spaces.

The most abstract view on a system can be taken, if it is considered a black box. That is the system's internals are invisible to the tester, except for service interfaces which allow to interact with the black box. These interfaces provide a set of methods that may accept or produce messages and can be used according to a protocol. Instead of examining the internal state of the system, only the observable external state, which is given by the service protocol, is examined [7]. To validate the system, it has to be checked whether the expectations on the interface interaction hold. This can be achieved, e.g., by running test cases. In order to face the SAS's behavioral complexity, these test cases do not have to be designed manually. The state space can automatically be searched for appropriate test cases, if the protocol is represented as a formal model. Test case generation from models is typically called Model-based Testing (MBT, [8]) and has been subject to recent research.

In general, a test case is a sequence of actions of two different types. Firstly, there are actions that produce certain messages to the system under test (SUT) in order to *enforce* (i.e., reproduce) a certain state of the protocol. Secondly, *assertions* retrieve messages from the system in order to verify their data against the expected values. Hence, this data has to be computed by a so-called test *oracle*, which is a mapping function from input to output data. Both, the values to be enforced and assertions to be validated, are central entities of a test model.

Before the test run, the environment of the SUT has to be properly set up. In order to validate SAS, changes have to be applied to this test environment such that self-adaptation is triggered. However, some of the environment's properties are not always controllable with acceptable effort. For instance, changing the weather in outdoor scenarios may only be feasible

by mocking sensor data. A related problem constitutes when the test model is not precise enough to predict exact reactions of the SUT. For instance, a robot that is controlled by an SAS does not drive very precisely to a predicted place. The decision logic of the SAS under test may recognize the drift and react properly, but the test oracle does not take the drift into account as it is based on the incomplete formal model. In summary, some validation steps may depend on information that is only available at test run-time and has to be gathered using sensors. For such situations, the validation mechanism has to keep a decision model in memory in order to adjust the validation process accordingly. In consequence, the test model is executed and adjusted at run-time, which is identical to simulation-based validation. The model is taken as a simulation input and its state is validated against the real system during execution.

Both methods, MBT and simulation, are based on a model that has to reflect the black box's external behavior. Like in SAS design, the test metamodel should be expressive enough for defining concise test models with a minimal effort. In [1], we have already investigated which properties of SAS are relevant for SAS testing. In order to provide model properties that hold for arbitrary SAS, we derived a general notion of such systems based on the concept of feedback loops that are commonly accepted in research as the central concept of all SAS [2]. In our original work, we extracted three different types of artifacts from this concept:

1) Failure scenarios, that can be employed for estimating the effort and progress of system validation.
2) Properties of failures for identifying meaningful verdicts (i.e, semantics of test results).
3) Potential error propagations and their causal chains.

The investigation process was based on *Failure Mode and Effects Analysis* (FMEA, [9]), a safety engineering method. By applying this sophisticated tool set, the analysis was founded on a solid methodology. In this paper, we extend the contributions of our original work as follows:

1) *Example*: In order to improve the understanding of the analysis process, we provide a domestic home robot application and illustrate the single analysis steps based on this example.
2) *Non-functional properties:* In our original work, primarily functional criteria were considered. In this paper, we additionally discuss the challenges originating non-functional criteria.
3) *Abstract reference solution scheme:* Based on the identified requirements, we propose methods and a reference solution scheme of interconnected artifacts that separate the minimal requirements of imaginable solution metamodels.

Besides these new contributions, we enrich our explanations with additional details.

The remainder of this paper is structured as follows: We start with related work in Section II. In Section III, we present the example SAS. In Section IV, we recite and extend our FMEA-based investigation for SAS by non-functional criteria and in Section V, we state the resulting modeling requirements. In Section VI, we propose the solution scheme. Finally, in Section VII, we outline future work.

II. Related Work

In literature, related approaches concerning testing adaptive systems have been discussed in two different research directions. Firstly, in context-aware and context-adaptive system research, model-driven test approaches were found that derive test data from context models. Secondly, several research groups developed methods around SAS engineering.

The most advanced approach concerning context adaptivity is, w.r.t. our knowledge, the work of Wang et al. [10]. The authors propose to construct an abstract control flow graph (CFG) directly from code artifacts by searching for access instructions on data that was delivered from a context middleware. Thus, a grey box perspective is taken, where at least the points of interest that rely on context data are identified throughout the source code. The CFG is an operational test model consisting of transitions and nodes (so-called *context-aware program points*, capps) that point to program locations. From the CFG, sequences of capps are generated. A context manipulator component generates sequences of context manipulations that are applied to the real system. In advance, the system code has been instrumented with feedback instructions that let the context manipulator monitor whether it triggers a certain capp. The sequences of context manipulations are optimized in order to trigger new states in the generated capp sequences. If they do so, a new test case can be assembled from the context manipulating actions. Using Wang et al.'s methodology, relevant operational orders of context manipulations can be automatically derived. Furthermore, their impact on the system can be identified such that a causal link between environment data and adaptation can be defined. However, the rest of the adaptation remains unconsidered. There are no means to validate whether any adaptation outcome is correct.

In SAS research, the earliest statements on the necessity of testing were published by Cheng et al. in [6]. The authors proposed to focus on adaptive requirements engineering and run-time validation to assure SAS's quality. However, they also constructed an abstract model of adaptive software's states consisting of an inner system state plus an adaptation mode or phase. The latter one describes in which variant a system works. Each transition, concerning either mode or state, changes the overall configuration of the system and has to maintain certain local or global properties. It is also discussed that a steady model as behavioral specification is insufficient for a behavior specification of SAS. While the authors' proposals are general enough to abstract from specific self-adaptive systems, several problems remain. Due to the enormous complexity in the behavioral space of adaptive software, an exact limitation of possible transitions to those which are correct and relevant for testing is a very hard task. In consequence, a much more expressive and usable model should be applied in test modeling.

A concrete research project on self-adaptivity is DiVA (Dynamic Variability in complex Adaptive systems). Despite comprehensive findings on engineering SAS, it includes a methodology for testing [11][12]. DiVA's validation process is split into two phases: (1) The early validation is based on design time models (adaptation logic and context model) and executed as a simulation. A main focus in DiVA's test method is to generate reasonable context instances and associate "partial" solutions, which can be used to find a set of valid configurations. (2) Additionally, an operational validation method is proposed that also deals with context changes/transitions. Therefore,

DiVA uses Multi-dimensional Covering Arrays (MDCA) including a temporal dimension. These arrays describe multiple context instances that are scheduled as test sequences and provide means for defining coverage criteria on sequences of adaptations. There are also fitness functions that help to minimize the test cases while preserving a good coverage. A drawback of this approach is that the test oracle is manual and does not depend on the previous configuration of the tested SAS. The latter point makes it impossible to examine stateful adaptations where not every system variant can be reached in arbitrary situations.

In context of DiVA, Munoz and Baudry presented in [13] an extended approach that bases on the same workflow. The authors formalize context and variant models and generate sequences of context instances by using Artificial Shaking Table Testing (ASTT). In order to measure the similarity between two context instances, a difference function is defined. Using the means of statistical distribution, context sequences are then generated, which are optimized towards a specific distribution of differences throughout the sequences. The theory behind this work states that sequences with at least one violent peek of difference between three following context instances would be best for testing the SAS. The theory was examined by showing an improved number of found failures in comparison to "non-violent" sequences. The ASTT approach is more powerful than the original DiVA test method as the oracle is defined formally and its predicted system variants also depend on the previous system configuration. However, the drawback of both DiVA test concepts is the lack of a method to validate behavioral adaptation.

Remedy to this lack give Abeywickrama et al. with their *State Of The Affairs (SOTA)* modeling and simulation platform [14]. They propose to model the complete behavior of SAS by specifying feedback loops as first class entities in form of activity diagrams. The feedback loops may communicate by hierarchy, shared components, or events such that different aspects of the system can be separated properly. Despite its modeling capabilities, a simulator, called SimSOTA, was developed that allows the live execution of SOTA models and their inspection during this execution. While the approach allows the definition of parametrical adaptation (using variable assignments) and behavioral adaptation, there are no means to predefine environment change. Thus, no system state can be automatically reproduced and the tester relies on external mechanisms for this purpose. In consequence, SimSOTA is more appropriate to be used in debugging a system in order to observe inconsistent states.

Another promising early-state work has been proposed by Nehring and Liggesmeyer in [15]. The approach enables an SAS engineer to inspect the system's state space exploratory. The authors assume that the system is component-based and the adaptation is a reconfiguration of this component structure. Along six examination iterations, transactional reconfigurations are investigated with a grey-box knowledge on the components and their interconnections. During the first iteration, a system model is constructed and workloads are defined to stress the system to examine its reaction to different situations. In the second iteration, the structural changes are checked. A third iteration is run in order to analyze whether data integrity is violated when different workloads are applied. During the fourth iteration, correctness of state transfers between exchanged components is examined. In order to evaluate the correctness

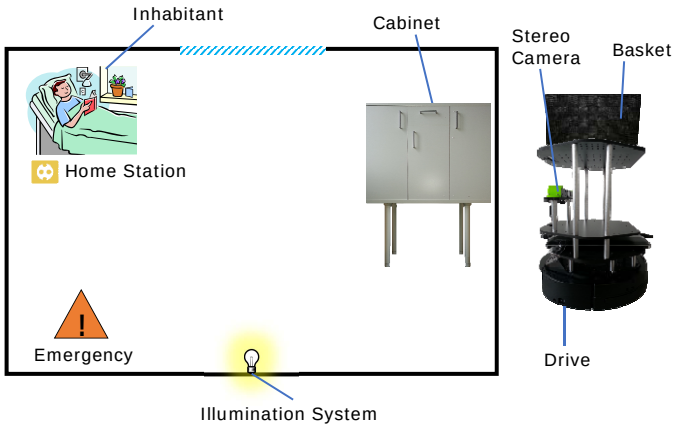


Fig. 1: The HomeTurtle application.

of transactions that are caused by adaptations, a fifth iteration is run. The sixth and last iteration has the purpose of checking identity relations of components before and after adaptation. As SOTA, this approach can be seen as a special form of debugging without means for environment situation enforcement.

In summary, existing approaches either lack the consideration of behavioral adaptation or they are not capable of enforcing a certain environment or adaptation state, which would be necessary for a systematic validation. Furthermore, non of the proposals has explicit means for tackling non-functional properties. However, any potentially complete test approach for SAS should be able to deal with non-functional requirements as well.

III. Example Application

In order to illustrate the analysis and solution scheme in the remainder of this paper, we sketch an example SAS in the following. We built the domestic home robot system *HomeTurtle*, which supports a disabled person at home. An example scenario is depicted in Figure 1. The service aims to deliver requested items to the inhabitant from a software-controlled storage cabinet. The scenario involves three active elements:

- *Transport Robot:* An extension of the TurtleBot platform (`http://www.turtlebot.com`) operates as transportation system. On top of the mobile robot, a basket is mounted where items can be put in. The system includes an autonomic computing unit and a stereo camera such that it is able to locate itself.
- *Storage Cabinet:* The cabinet is controlled by a WiFI-connected embedded device. This device is capable of triggering magnetically hold flaps, which lock boxes each containing an item. After opening a flap, the item drops out and falls through the cabinet's base into the robot's basket. The robot's battery is charged when it parks on the its home station.
- *Illumination System:* If the natural illumination is insufficient for the robot to locate itself, a lamp can be switched on automatically. For this purpose, the Philips Hue is used (`http://www.meethue.com`). The bulb contains a WiFi-connected embedded control device as well.

In order to start the interaction, the inhabitant requests a specific desired item (by speech input) and the robot starts driving automatically. It plans its way through the room and avoids obstacles that are recognized by using its stereo camera. After finding the cabinet, the robot parks thereunder. By WiFi connection the cabinet's embedded device is signaled to drop the requested item. Then, the robot drives back to the inhabitant for delivery. In a final step, the robot drives on its home station where its battery is charged as long as no request is being operated.

All decisions are made autonomously by an SAS. The computations take place on the computation unit hosted on the robot. By adding several self-adaptive capabilities, this software allows the robot to work in different situations effectively. Firstly, the correct recognition of walls and obstacles relies on appropriate room illumination. When the natural brightness (through the windows) is insufficient, the system automatically switches on the illumination system to improve the quality of obstacle detection. Secondly, the inhabitant can use an emergency switch. If this switch is triggered, the robot is expected to immediately cancel its current action and navigate to an emergency location as labeled in the figure. The purpose of this operation is to avoid the robot being an obstacle while human emergency responders are in the room. Thereby, the moving robot would be a danger itself.

We implemented the HomeTurtle system to experiment with our adaptive software framework *Smart Application Grids* (SMAGs) [16]. Applications that are built using SMAGs are able to adapt their component-based architecture. For example, component implementations are exchanged or components are automatically connected by generated adapters at run-time. Based on these features, we have built the above described home robot system. As the decision logic of the HomeTurtle bases on the reflection of physical objects and actuators, the system can be categorized as CPS. In the remainder of this paper, the capabilities of the HomeTurtle are used to illustrate our analysis process as well as the solution scheme proposal.

IV. Failure Analysis

In this section, we analyze relevant failure characteristics and scenarios of SAS. For this purpose, we apply FMEA [9][17]. FMEA is used in engineering of safety-critical systems to find relevant failure sources. The method was first applied for electrical and mechanical systems and later extended for the usage in software engineering [18][19].

According to [7], a *failure* is an event of service deviation from an expectation. The expectation is defined in a specification document, e.g., in form of requirements. An *error* is the inconsistent part of the total system state (internal state plus perceivable external state) which may *propagate* the failure. The cause that *activates* an error is a *fault* that is *active*. There may also be *dormant* faults, which do not cause errors. A failure may trigger a new fault in another component as well. This interaction is called *causation*.

Based on the FMEA process, our analysis is separated into three steps:

1) Identification of SAS-specific failure dimensions and properties (presented as *Failure Domain Model*).
2) Investigation of SAS-specific failure scenarios.
3) Visualization of error propagation among the found scenarios as *Fault Dependency Graph*.

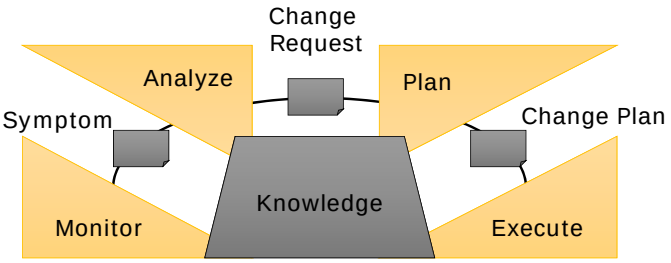


Fig. 2: The MAPE-K feedback loop (cf. [2]).

Step (3) is not an actual part of FMEA. Usually, a *Fault Tree Analysis* (FTA) [20] is performed to visualize the scenarios' dependencies and to enable engineers to trace which faults may have caused a certain failure. The result of FTA is a Fault Tree Set (FTS) comprising multiple trees that represent how a fault may be propagated through the system. Because SAS run a feedback *loop*, error propagation in SAS cannot be described in the form of a tree in general. Hence, we customize the analysis process in this step by constructing a directed graph with logical gates instead.

A. A Common Process of Self-Adaptation: MAPE-K

Before starting the analysis, the level of detail has to be specified in order to set up a fixed abstraction perspective on SAS including a well-defined system boundary. FMEA is designed to be ran against an existing technical architecture, which we cannot generally assume to be widely similar in all existing or future developed SAS. Hence, we discard the strict understanding of FMEA by analyzing a general conceptional architecture that comprises minimal necessary components and data flows in between. As seen in the previous section, there are several intersecting research directions coping with self-adaptivity. They have in common, that the process of information gathering and utilization relies on the feedback loop principle of autonomous systems. The steps that are performed during the execution of this loop are (1) Monitoring of sensor values, (2) Analyzing whether adaptation is required, (3) Planning the adaptation and (4) Executing the plan. During these phases, internal or external Knowledge sources can be used to retrieve or store information relevant to the decision mechanism. This process concept is called the *MAPE-K* feedback loop [2].

As illustrated in Figure 2, the phases of MAPE-K exchange multiple information entities. The system monitors a set of data sources such as *sensors* for external entities or *system interfaces* for internal properties. In our HomeTurtle SAS, the monitored data encompasses a brightness value (detected by the stereo camera) as well as a WiFi-retrieved temperature signal. The set of environment and system states is the relevant computation base for all later decisions–it assembles the *context* of the SAS. The captured information is then inferred to symbolic situation specifications, called *symptoms*. The HomeTurtle software maps the concrete brightness values to symbolic values like `Illuminated`/`Too_Dark` and produces a symptom `Emergency`, if the temperature exceeds a certain level. The symptoms are forwarded to the analysis phase where the system reasons about the necessity of adaptation. Therefore, the conditions of adaptation policies are compared with the symptoms of the current situation. The HomeTurtle's policies are based on Event-Condition-Action (ECA) rules. For instance,

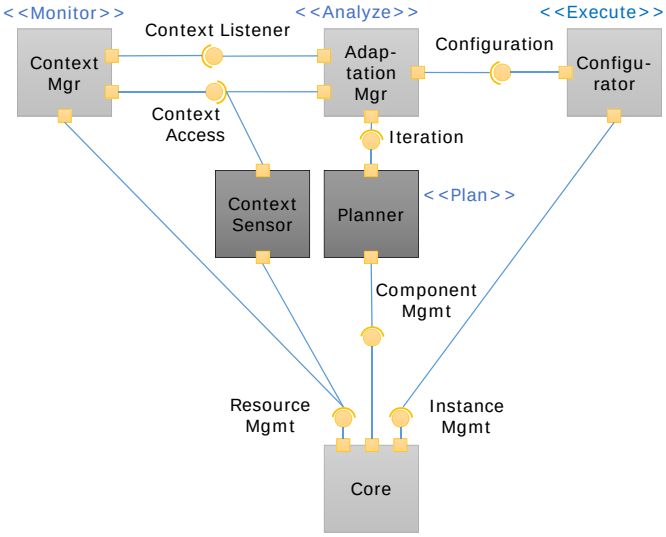


Fig. 3: Example SAS architecture according to [21].

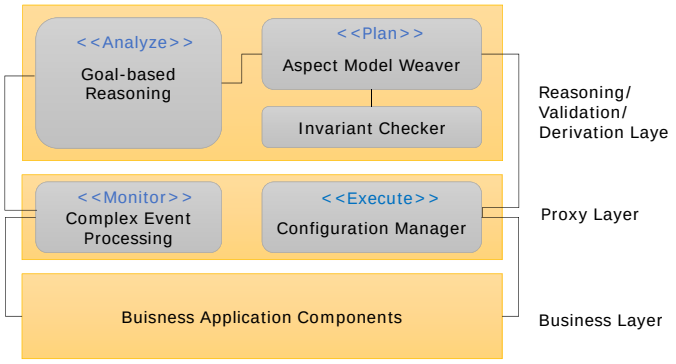


Fig. 4: Example SAS architecture of DiVA according to [22].

there is a rule that is triggered by the emergency signal and commands the robot to perform an emergency adaptation.

Up to this point, the system has determined *if* an adaption should be performed, which is signaled by a *change request*. During the subsequent plan phase, a *change plan* is generated. This plan comprises an operational definition of adaptation actions. An action defines how components of the SAS are changed in detail (e.g., by setting new parameter values or by re-composing the system from modules). In case of the HomeTurtle, the action is to cancel the current process and to redirect the robot to the emergency position using the navigation components. Subsequently, during the execute phase, the plan is applied. This may also involve *effectors* manipulating external entities of the environment. The whole feedback loop is re-ran from this point periodically such that a self-adaptive system always has the intended state (e.g., according to the supposed utility function or goal, and available knowledge) to fulfill its task.

Any SAS architecture adheres to variants of this feedback loop. For instance, Hallsteinsen et al. proposed in [21] a platform based on dynamic product line techniques as depicted in Figure 3. The `Context Manager` component can directly be associated with the monitor phase as it collects and reasons about information that were gathered from resources, the environment (by sensors), or humans. The `Adaptation Manager` then decides whether an adaptation is required based on the context changes and, thus, implements the analyze phase. The `Planner` is responsible of generating a plan for reconfiguring several variation points. Based on this plan, the `Configurator` applies the reconfigurations with the help of the core's instance management interface.

Another example is the DiVA [22] project, whose architecture is depicted in Figure 4. On the lowest `Business Layer` the architectural model of the managed applications is hosted. It contains probes that generate run-time events. These events are consumed by a `Complex Event Processing` component on the `Proxy Layer`. The events are monitored, interpreted, formatted and a context model is updated monitor phase. This context model is input to the `Goal-based Reasoning` component on the upper level which decides whether adaptation is required (analyze phase). As the adaptive capability of the DiVA approach is based on Aspect Weaving, the respective component plans the adaptation. During the plan phase, an `Invariant Checker` determines if the result of weaving fulfills a set of given constraints. Finally, the weaving plan is directed to the `Configuration Manager` that applies the changes to the business application (execute phase).

As the approaches of Hallsteinsen et al. and DiVA, components of arbitrary SAS can be mapped to the feedback loop principle. MAPE-K encompasses the adaptation according to environment changes such as context, user input and system utilization. The sources of processed information can be abstracted by leaving out the concrete objects, which are observed or controlled by sensors and effectors respectively. In the following analysis, we use MAPE-K as common viewpoint on SAS architectures. We assume engineers, who are in charge of validating the adaptation capabilities of the system, are equipped with the required tooling to observe the data exchange between the loop's phases.

B. Step 1) Failure Domain Model (FDM)

In the following, we provide a set of generic failure properties for SAS. As there are several classes of SAS [23], we cannot assume that each property is reasonable in every concrete system. Instead, the proposed properties are a superset of properties that can occur in MAPE-K-based systems on the discussed abstraction level. Furthermore, we exclude failures that originate in sensors and effectors to create a fixed boundary around the software's scope.

As briefly discussed, each component of an SAS provides services through its interface. In the system's requirement specification the expected behavior of these services is defined. Notably, the specification comprises functional and non-functional requirements. Thus, besides concrete features, which have to be supported by the SAS, the quality of how the features work is constrained. In the HomeTurtle example, a functional requirement is the ability of the robot to navigate through the room without colliding with any obstacles. Associated non-functional requirements are, for example, a lower bound for the precision of the obstacle detection (i.e., percentage of correctly detected obstacles) and upper bounds for the total time required by the robot to navigate from the inhabitant to the cabinet and back.

Basically, the definition of faults, errors and failures [7]

holds for arbitrary systems. However, in self-adaptive applications, the boundary between faults and errors can become blurry. Consider, for example, a system that uses models at runtime [24], where the current system state is kept and abstracted in a model to specify the adaptation logic as decision rules against this model. This system is, in principle, able to adapt these rules such that the adaptation logic of the system changes. If this adaptation has not been performed correctly, a new fault is introduced that was not created at design time but at runtime. In other words, due to the ability to dynamically change the system specification, failures can create new faults at runtime. Thus, for adaptive systems which are able to manipulate their own decision logic, faults and errors may not always be distinguishable.

In [7], for each fault, error, and failure a comprehensive list of property dimensions is given. However, as we already defined the level of abstraction and only consider elements that are generic to arbitrary implementations of the feedback loop, this list can be filtered. For instance, it includes properties that specify severity, the cause why, or the life cycle phase when a fault was created. In black box testing against requirements, these information cannot be evaluated as they can neither be observed nor deduced. Concerning failures, for instance, detectability is not relevant in black box testing as non-detectable failures can never be found due to the lacking knowledge on dormant faults. Furthermore, when comparing different notions of FMEA (e.g., [7] and [18]) the FDMs (that are assumed to be generic) differ. In consequence to these issues, we designed our own FDM that only proposes properties that are relevant to SAS in particular.

The resulting FDM for SAS is depicted in Figure 5. Concerning faults, the only property that can be deduced from observed behavior is their *persistence* which may either be *permanent* or *transient*. Detecting permanent faults is usually less challenging than detecting transient faults. For instance, the HomeTurtle aggregates the last hundred temperature values in order to compute their average and to decide whether a fire alert has to be signaled. If the collected values are not deleted after an appropriate time period, the system may run into a memory overflow and stops working permanently. In contrast, when only an outlier value distorts the current value queue, this fault vanishes after a certain time period and is transient to an observer.

Regarding errors, we propose the dimensions *type* and *localization*. In our black box abstraction, the engineer is able to observe whether a false state establishes in the inner knowledge model of the system or in the computational process. For the first type, two deviations are possible: either the model does not correctly reflect its subject (e.g., the stored location of the transport robot is not correct) or its inner constraints are incorrect (e.g., more brightness values are stored than expected). The HomeTurtle may also have process-related errors, for instance, when the emergency symptom is erroneously produced and the system runs an unnecessary adaptation. Furthermore, errors can localize either locally or globally in the potentially distributed SAS. In our example, the set-up of items in the cabinet could not reflect its real contents. This error would be shared between all technical elements over the WiFI network.

Concerning failures, the authors of [7] distinguish between content and timing (early/late) correctness. In contrast, an SAS's goal definition may aim on other non-functional properties like energy-usage. Thus, we alter the original distinction to *non-functional* (i.e., the service performs not with the expected quality) or *functional* (i.e., incorrect service behavior). In our example, we can distinguish between failures where the robot does not deliver an item within a required time or completely fails in delivering it. Furthermore, failures in SAS can either manifest *internally* (e.g., in an inconsistent model) or *externally* (e.g., when the robot heavily collides with an obstacle). Concerning, the *Appearance* dimension, sensed and analyzed information may lead to un-intentional (*false-positive*), missed (*false-negative*), or *semantically* wrong sensor events, change request, or adaptation actions. The HomeTurtle, for instance, may start driving to the emergency position without cause, it may miss the emergency signal or mis-interpret it.

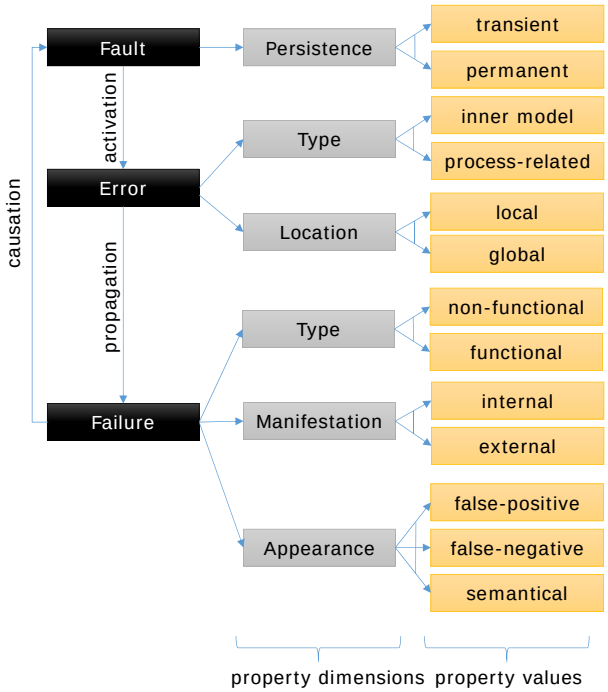


Fig. 5: The SAS Failure Domain Model.

This FDM is a valuable source for classifying faults, errors, and failures in concrete SAS. It describes their abstract properties that can be instantiated for real world systems. Verdicts (i.e., the classification of test results, for instance `Pass`, `Fail` or `Inconclusive`), can be parametrized by these information.

C. Step 2) Failure Scenarios

In this section, we identify scenarios of failure occurrence in SAS. In contrast to other FMEA applications, we cannot give a general method of prioritizing these scenarios as this strongly depends on domain-specific conditions. The only applicable, general evaluation standard is *criticality*. In this case, according to [23], each adaptation operation can be either harmless, mission-critical, or safety-critical. When the HomeTurtle robot drives along an non-optimal path, this failure is harmless. In contrast, not delivering an item would be mission-critical. If the robot collides heavily with a human being, the failure can even be safety-critical. However, other SAS may have completely different requirements such that statements about the severity of failure scenarios cannot be generalized.

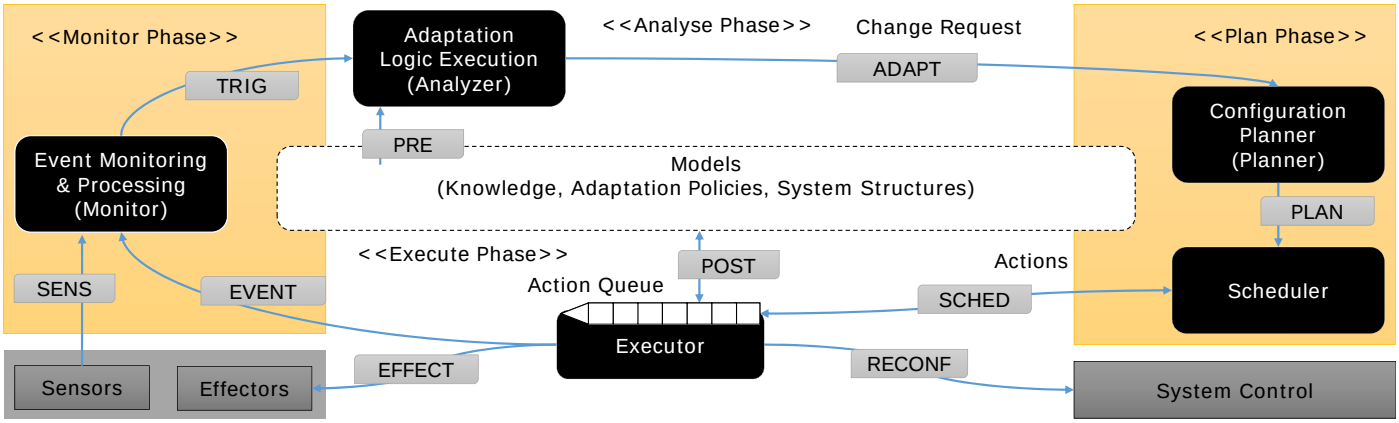


Fig. 6: Conceptional architecture of SAS.

TABLE I: SAS failure scenarios.

FID	**CID**	**Fault** in...	**Error** in...	**Failure** in...	**Propagation**
SENS	Monitor	sensor interpretation ► transient or permanent	environment reflection ► process-related ► local or global	produced symptoms ► functional or non-functional ► internal ► false-positive, false-negative or semantical	TRIG
TRIG	Analyzer	symptom interpretation ► transient or permanent	adaptation decision ► process-related ► local or global	change request ► functional or non-functional ► internal ► false-positive, false-negative or semantical	PRE/ADAPT
PRE	Analyzer	model interpretation ► transient or permanent	adaptation decision ► process-related ► local or global	change request ► functional or non-functional ► internal ► false-positive, false-negative or semantical	TRIG/ADAPT
ADAPT	Analyzer	reasoning algorithm ► transient or permanent	adaptation decision ► process-related ► local or global	change request ► functional or non-functional ► internal ► false-positive, false-negative or semantical	PLAN
PLAN	Planer	planning algorithm ► transient or permanent	planning decisions ► process-related ► local or global	change plan ► functional or non-functional ► internal ► semantical	SCHED
SCHED	Scheduler	scheduling algorithm ► transient or permanent	scheduling decisions ► process-related ► local or global	wrong order of actions ► functional or non-functional ► internal ► semantical	POST/EVENT/ EFFECT/RECONF
POST	Executor	model manipulation ► transient or permanent	model ► inner model ► local or global	model inconsistent ► functional or non-functional ► internal ► semantical	PRE/PLAN
RECONF	Executor	reconfiguration ► transient or permanent	system configuration ► inner model ► local or global	system reflection ► functional or non-functional ► internal ► semantical	–
EVENT	Monitor	system event monitoring ► transient or permanent	system reflection ► process-related ► local or global	system events ► functional or non-functional ► internal ► false-positive, false-negative or semantical	TRIG
EFFECT	Executor	actual control ► transient or permanent	erroneous actuator commands ► process-related ► global	environment state ► functional or non-functional ► external ► false-positive, false-negative or semantical	(SENS)

The scenario identification relies on the feedback loop's conceptional structure, which is depicted in Figure 6. Hence, we decompose the MAPE-K loop in five components: `Monitor`, `Analyzer`, `Planner`, `Scheduler`, and `Executor`. The latter two are separated, to enable the consideration of *interaction* between adaptation and running processes. After the `Analyzer` detected *that* the system has to be adapted, the `Planner` decides *how* the adaptation is processed. The `Scheduler` has the task to fill an `Action Queue` (however, it may be implemented in concrete systems) by arranging system process actions with adaptation actions. An adaptive system designer has to be aware of how he maintains consistency either through an actual implemented scheduler component or a transaction-like behavior. This issue also breaks the straight MAPE-K data flow because a scheduler requires information about the current system actions (retrieved from the `Executor`) and composes them with adaptation intents. In the HomeTurtle system, this feature plays an important role as well. For instance, when an emergency is signaled, the delivery process is expected to cancel immediately. In contrast, an adaptation concerning illumination conditions would make no sense while the robot is parked under the cabinet. In this case, the scheduling implementation is expected to first atomically execute the waiting operation and run the adaptation afterwards.

All components are considered as black boxes and are connected by data flow edges (blue arrows). Additionally, the process contains `Sensors`, `Effectors`, `System Control`, and the central knowledge `Models`. The latter one contain information about the system structure, adaptation logic, and further knowledge relevant to adaptation decisions. `Sensors` and `Effectors` communicate with the external world (e.g., other systems or the physical reality). `System Control` provides an interface for system reconfiguration actions that are controlled by the `Executor`.

Based on this structure, we derive failure scenarios by investigating potential wrong processing of input data by a certain component. As there may be multiple outputs of a component, each component can produce multiple failures. We list our found failure scenarios in a worksheet as presented in Table I. A scenario represents the possible occurrence of a fault and its related causality chain. The description of each scenario comprises *Failure Identifier* (FID), *Component Identifier* (CID), Fault, Error, Failure, and a *Propagation* column. All FIDs can be found in the architecture visualization in Figure 6. In the following, each scenario is described in detail. As an extension to our original work, we add examples and discuss non-functional aspects as well.

SENS: The first scenario comprises test input received from the `Sensors` and misinterpreted by the `Monitor` component. During the interpretation, values are mapped, aggregated and inferred. It might also be the case, that a history of values is maintained in order to infer over them. Such a fault can activate an error that comprises an incorrect reflection of the environment such that the produced symptoms are incorrect.

Example: The HomeTurtle's monitoring component collects brightness values over the last ten minutes but fails to discretize them correctly. The symptom that indicates a low illumination is not being produced as expected.

Non-functional aspects: This scenario comprises test input received *too late* from the `Sensors`. Faults of this kind activate errors comprised of the reflection about outdated observations of the environment such that the produced symptoms are incorrect.

Example: The HomeTurtle's monitoring component collects brightness values only every minute. Thus, in the worst case, the HomeTurtle will not adjust itself to a changed illumination for almost a minute, which is not the expected behavior. A higher sampling rate has to be used.

TRIG: A symptom produced by the `Monitor` does not or unintentionally trigger a corresponding change request or a wrong one.

Example: The HomeTurtle's analysis components fails matching the symptom description with an adaptation policy's condition. Thus, the expected adaptation is not being performed as expected.

Non-functional aspects: A symptom produced by `Monitor` triggers a corresponding adaptation *too late*.

Example: The HomeTurtle observes its own remaining battery capacity. If a user requests the HomeTurtle to deliver an item from the cabinet, the HomeTurtle checks whether its remaining energy suffices to fulfill the request and will incorporate a stop at the charging station if the remaining capacity is too low. If the analysis' result reflecting whether the capacity suffices or not is send too late to the `Planner`, an erroneous plan is generated, which does not include the stop at the charging station.

PRE: The SAS knowledge resources contain information that may constitute pre-conditions to an adaptation decision. If these information are misinterpreted, the adaptation decision can differ from the designer's expectation.

Example: According to the HomeTurtle adaptation policies, the illumination system has to be switched on as soon as the obstacle recognition precision deceeds a certain level. Thus, the current precision is inferred using the empirical metrics on the stereo camera's precision that is parameterized with the currently detected brightness value. If the empirical model is corrupt, the adaptation of the illumination system is not performed in the relevant situations.

Both TRIG and PRE scenarios may interact, because we did not decompose the analyzer in more detailed components. Hence, these scenarios have to be tested together as both data sources are required for each test case and just a probabilistic estimation can be stated which one is actually defective.

ADAPT: Depending on the deduced adaptation decision, the system is in charge to produce and correctly interpret the change request. In cases where this output is corrupt, adaptations are not performed as expected.

Example: After the brightness value was found too low, the respective change request was derived but is not forwarded to the planner due to an exception. Thus, the adaptation initiation is lost.

PLAN: The `Analyzer` determines *if* an adaptation is required but *not how* to perform it. This task is operated in the planning phase. A `Planner` reasons over the variability and the current system state. Its output has to be a correct adaptation plan that can be applied in the system and leads to a consistent state. The PLAN scenario encompasses that the compiled plan is incorrect.

Example: In the emergency case, in the constructed adaptation plan the canceling of the current delivery task and is queued after the actions necessary for driving to the emergency position. Thus, the robot first delivers the item and approaches the emergency position subsequently, which was not intended.

Non-functional aspects: Often, the `Planner` reasons over non-functional properties like response time or energy con-

sumption. But, the planning task itself effects these properties by utilizing the same resources. In consequence, the resulting decision of the `Planner` is infringed, because the assumptions taken by the `Planner` are violated.

Example: The HomeTurtle shall drive to its charging station if the battery capacity falls below 10%. To reach the charging station, the HomeTurtle consumes energy. But, executing the planner, to decide whether to drive to the charging station or not, consumes energy, too. Thus, by executing the planner, the maximum distance of the HomeTurtle to its charging station is decreased. If the planner does not consider this decreased distance, the decision to go for charging, will be made too late.

SCHED: Reconfiguration actions potentially interact with the system's control flow. Such problems arise because variability cannot be completely orthogonal to the system's task execution. The expectations of the designer may even encompass that certain actions may be transactional. Differing from these expectations activate errors in the scheduling process and are observable as wrongly ordered actions.

Example: Due to a wrongly designed scheduling component, new adaptations actions are enqueued behind all previously initiated system actions. Thus, for instance, the illumination adaptation is being deduced while driving to a certain position but performed after the driving process. Then it may be too late to avoid a collision.

The `Executor` is a complex interpretation engine that produces multiple outputs and thus, has multiple potential failure scenarios. All `Executor`-related scenarios may also be the outcome of a propagated SCHED failure.

RECONF: The reconfiguration may run into a failure itself. If any reconfiguration mechanism fails without being recognized, the actual system structure is out of synchronization with its model representation.

Example: The driver for the control of the robot's wheels is implemented as a blocking service such that the emergency adaptation cannot be performed immediately. Instead, the designer expects the system to cancel the operation.

POST: The `Model`'s part that represents the reconfigured systems may be inconsistent after the execution because a model manipulation was performed erroneously by the `Executor`. Thus, the reflected system state may differ from its real configuration. This deviation may harm future adaptation decisions in form of wrong preconditions (PRE). The failure can be observed in the model's state.

Example: The illumination adaptation is correctly performed but due to an exception this change is not reflected in the model. During the next adaptation loop, the decisions that depend on this information will be erroneous.

EFFECT: Another output of the `Executor` can be actions that have to be performed by external systems using the `Effectors`. If actions are not generated correctly and forwarded to the effectors (e.g., due to corrupt drivers), the representation of these externals loose synchronization with potential internal model representations. As the `Sensors` may perceive data from the manipulated system, a SENS scenario may be caused indirectly.

Example: The signal to the illumination system may be misinterpreted and the Hue bulb is not switched. While the physical condition remains unchanged, the SAS now assumes the environment to be altered as it reflects its own actions in the knowledge model.

EVENT: The last failure scenario is related to events that are produced in the software system and are propagated to the `Monitor` component which makes them part of the context representation. In the related failure scenario, the generated events are erroneous.

Example: In order to avoid concurrent adaptations, the HomeTurtle is expected only to obtain new sensor data when all adaptation actions are finished. Afterwards, a respective event is produced by software that causes a re-start of the feedback loop. Loosing this trigger event constitutes a failure.

Properties of faults, errors, and failures as proposed in our FDM can depend on the concrete architecture of the system. Based on the assumption that the system was built as our abstract feedback loop suggests, some property values can be neglected. In our scenarios worksheet, the remaining ranges are denoted. Regarding faults such a restriction cannot be deduced from the general feedback loop. Thus, arbitrary SAS can include transient or permanent faults in each of the proposed components.

Concerning errors, only the `Executor` is expected to directly manipulate the model. Thus, the inner-model error type only occurs in this component when reality is no longer correctly reflected in the system's knowledge base. All other faults impact to process-related state of the system. Depending on whether the SAS is distributed or not, each component may propagate its potentially erroneous outputs through the complete infrastructure. However, effects that propagate out of the system always have to be considered global, as all monitoring components may detect external changes.

The appearance of a failure depends on the content of a component's output. In cases of decisions (symptoms, change requests, system events, effector actions), they may be missed (false-negative), unintentionally performed (false-positive) or semantical wrong. In cases of adaptation plans and model manipulations, which are always expected to be produced, the potential errors impact their contents only. All components may produce such functional failures. Despite the non-functional aspects which were discussed in context of the `SENS`, `TRIG`, and `PLAN` scenario, each computation can be limited in its budget usage in general. Thus, we propose to consider non-functional failures in all scenarios.

D. Step 3) Fault Dependency Graph

As final artifact, we construct a faullt dependency graph as depicted in Figure 7. The nodes of the graph identify each a certain failure scenario or a logical or gate. Connections reflect the causations as listed in respective column of Table I. The visualization illustrates the potential cyclic failure propagation through inner system events, model manipulation, or physical sensors or effectors correlations (the latter one is visualized by the dashed edge). Furthermore the PRE and TRIG scenarios may influence each other in both directions, which makes them hard to test in isolation.

The graph can be used by engineers to identify potential sources of observed failures. Furthermore, quality assurance can be steered using the graph by measuring or estimating each scenario's probability of occurrence. Thus, it can even be deduced how probable a certain causal chain or how severe its a fault's impact is.

V. Requirements to Models for SAS Testing

All following requirements for self-adaptation test methods are based on a selection of the presented failure scenarios. In the

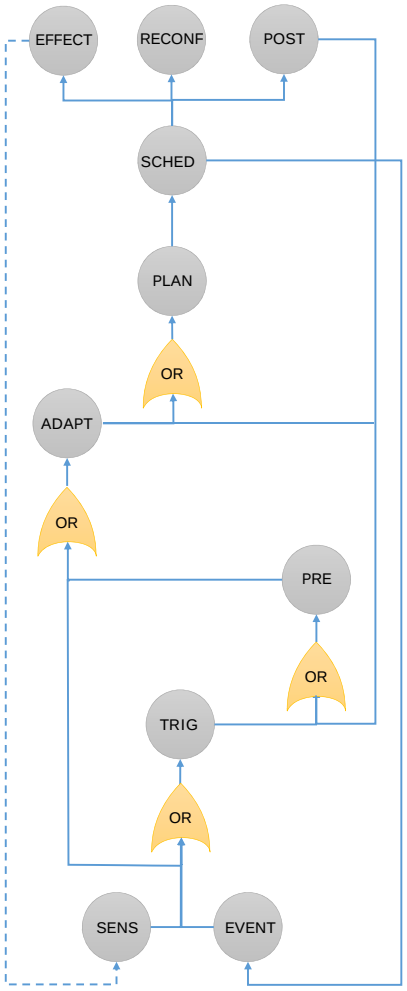


Fig. 7: Fault Dependency Graph.

following, we list these mapped requirements and name each of them for later reference. The requirements are formulated as *assurance tasks* that have to be fulfilled by employing validation methods.

A. Functional Requirements in SAS Testing

F1) **Correct sensor interpretation:** Assure that the sensor data is correctly interpreted and transformed into system events. Potential sensor data has to be specified together with context identifications. (↦SENS)

F2) **Correct adaptation initiation:** Assure that events initiate the correct adaptation if all preconditions in the model hold. Events, conditions, and adaptation decisions have to be associated in the models. (↦TRIG/PRE/ADAPT)

F3) **Correct adaptation planning:** Assure that the generated adaptation plan is consistent w.r.t. target configuration and action order. Build a model to map adaptation goals to possible plans.(↦PLAN)

F4) **Consistent interaction between adaptation and system behavior:** Assure that the generated adaptation plan is correctly scheduled with the system's control flow. A model is required to define which adaptation is allowed in which state of application control. (↦SCHED)

F5) **Consistent adaptation execution:** Assure that (1) the generated adaptation schedule is applied to system structure and (2) the synchronization between the running system and the models is consistent after adaptation. (↦POST/RECONF)

F6) **Correct system behavior:** Assure that the system correctly commits events or actions to the effectors. As in the previous requirement, here we need to specify events to be observed in the system when running any operation. (↦EVENT/EFFECT)

B. Non-functional Requirements in SAS Testing

Whereas the requirements to SAS testing in the last subsection focused on assurance of the SAS's functionality, a second type of requirements to SAS testing exists: non-functional requirements. The central concept for non-functional testing of SAS is the budget, which covers a boundary for a selected non-functional property. For example, the limited capacity of a battery sets a hard budget in terms of energy, which must not be exceeded. The following aspects of SAS require the consideration of non-functional requirements:

NF1) **In-budget sensor interpretation**: A failure due to incorrect sensor interpretation has a non-functional dimension in that, e.g., the timing behavior of the sensor interpretation is faulty. In other words, sensor interpretation has non-functional requirements, which must not be violated. These requirements include the ability to handle imprecise sensor data (precision), timing constraints on when and how long sensor data is valid and resource budget constraints, if the sensors are power by a battery, which is characterized by a limited energy capacity. (↦SENS)

NF2) **In-budget adaptation initiation**: In addition to the assurance of the correct adaptation to be initiated, it is important to ensure that this adaptation is initiated in time. If an adaptation is initiated too late, multiple qualities are potentially infringed. Imagine, for example, a self-adaptive system optimizing for energy efficiency. The later an adaptation to save energy is initiated, the more energy is consumed in the meantime, which infringes the goal of the self-adaptive system. (↦TRIG)

NF3) **In-budget adaptation planning and execution**: Also for planning, non-functional requirements have to be fulfilled besides correctness. Again, budgets of non-functional properties must not be exceeded. A particular problem in this regard is the assessment of the planning step itself in terms of various non-functional properties. For example, the runtime of an optimizer using integer linear programming is double exponential in the worst case, but almost linear in the average case [25]. In addition, the determination of the size of the respective budgets is a highly complex task. This is because often the goal of the self-adaptive systems is to optimize for selected non-functional properties, but performing optimization (and adaptation) effects these non-functional properties. (↦PLAN)

C. Required Test Adequacy Criteria and Coverage Metrics

Additionally, in testing, adequacy criteria are required to restrict the tested behavioral space and, nevertheless, have a reasonable and meaningful test result. Furthermore, coverage metrics are used to compute in which degree the state space is covered by the generated or performed test cases. For less complex systems, many criteria for test adequacy and coverage metrics were found. Mostly, they refer to a graph representation like a state machine. Known criteria are statement, branch or path coverage. However, as we have seen, there is a complex set of requirements and aspects to be tested in the context of SAS. In consequence, we have to use multiple models which are more expressive then state machines (as assumed in the mentioned coverage criteria) to represent all testable aspects. In consequence, the known criteria cannot be applied directly. Hence, the last requirement is to find a set of proper adequacy criteria and coverage metrics for SAS which can be composed:

C) **Adequacy criteria and coverage metrics for SAS:** Find constructive adequacy criteria metamodels/languages to describe *which*, *when* (in relation to system behavior), and *in which order* adaptation scenarios have to be tested and analytic coverage metrics for measuring a test suite or its execution.

VI. Reference Solution Scheme

In order to help testers facing the challenges stated above, we propose a scheme consisting of methods of fault detection and representation artifacts.

A. Considerable Methods

In black box testing, the SUT is represented by well-defined interfaces without defining its internal behavior. In our analysis, we based on the most abstract definition of an SAS, namely the MAPE-K feedback loop. We have derived five crucial components that are black boxes and provide interfaces where data can be sent to or received from. Despite the assumed informational entities defined by the knowledge element of the feedback loop, the components' internal state may effect the outcome of their computations. Thus, the black box interface can have a stateful protocol as well.

In order to validate the correct outputs of each component, several input data scenarios have to be specified and the output has to be predicted. Such test cases *enforce* a certain state of the interface protocol that is relevant during the prediction. In testing, the prediction task is solved by oracles. An oracle has to be specified according to the requirements of the SUT. In order to automate the validation process as far as possible, a specification is most useful in a formal representation such as a model. Models can be employed either for generating test cases or by executing them directly. The latter method is identical to simulation-based validation. Hence, the simulation model is executed in parallel to the SUT and the simulation's propositions are frequently validated against the SUT.

Both generation and simulation have several pros and cons. In Figure 8, the relation between the two methods is depicted. The first artifact, which is assembled during the system's development process, is the requirement specification. Based on this specification, the design is derived during the system's development process. The design models are refined in incremental steps until the code level is reached, which is the final representation. The refinement process may be be manual

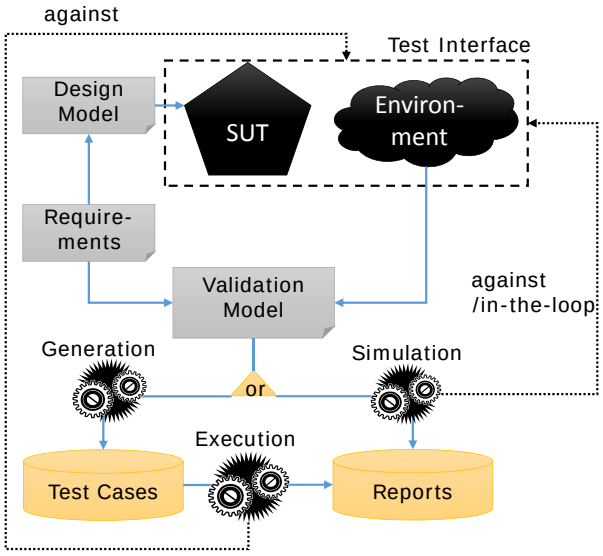


Fig. 8: Validation Methods.

or partially model-driven (i.e., Model-Driven Architecture). When the design model is specified, misinterpretations and mistakes in the design itself will create faults that later have to be uncovered by validation. Despite these fault sources, during the refinement the loss of information between two levels of abstraction may have effect on the system's correctness. Furthermore, due to the change of requirements during the whole process, inconsistencies of already implemented artifacts may arise. In the end, validation is responsible to examine how far the implementation still matches the requirements. Therefore, an engineer has to define a validation model according the initial requirements. The model should be independent of the system's design in order to avoid the re-implementation of erroneous interpretations. The validation model specifies inputs and assertion actions against the interface of the SUT. Additionally, it has to specify environment changes in order to trigger and validate the self-adaptation. Therefore, the environment's properties and the change of their values over time has to included by appropriate model representations.

The automatic generation of test cases from this model is called Model-based Testing (MBT) [8]. The generation process is controlled by an adequacy criterion, that specifies which entities of the model (i.e., states, transitions, data elements) have to be covered before the production of test cases terminates. A generated test case specifies a strict sequence of test actions that are applied against the SUT. All test cases are generated only a single time, as long the requirements stay fix as well. They are saved in a test suite and can be replayed over multiple regressions when the implementation has changed. During the test case execution, inputs are sent to a test interface and the resulting outputs are checked against the predicted data using *assertions*. The test interface provides appropriate access to system functions and environment control as well monitoring capabilities. For each run, it can be reported whether it completely succeeded or in which execution step a failure occurred. The MBT approach has two advantages. Firstly, all test cases are assumed to be strictly reproducible as their execution solely depends on the action that are sent to the SUT. Secondly, a coverage can be measured, e.g., by counting

the number of executed test cases in the relation to the complete test suite.

In contrast, simulation directly executes the model. Decision points during the model interpretation have to be made randomly or they have to be controlled by a user or a heuristic (similar to an adequacy criterion). During the simulation, failures can be observed in form of deviations from the real system's behavior. Each derivation can be stored in a report. During the simulation, data from SUT and environment can be queried and used to determine decisions. This principle is called "in-the-loop" and states the main advantage of simulation in comparison to MBT.

The decision between MBT and simulation depends on the existence of uncertainty-establishing artifacts that have to be involved in the loop. Both methods rely on a validation model, which has to implement the requirements stated in our previous analysis. Each of the defined requirements is a concern to this model. In the following section, we propose a conceptional concern-separated structure for such a validation model.

B. Counter Feedback Loop

The SAS deduce adaptations from monitored context changes. In contrast, a validation mechanism has to work exactly vice versa. In our work, we call this principle *Counter Feedback Loop* as depicted in Figure 9. The context has to be actively changed in order to trigger the SAS to adapt and enforce a certain adaptation state, whose effects can be examined. Hence, the test model has to include a `Change` specification containing a set of scenarios. Executing such scenarios allows to check whether sensor data is correctly obtained and inferred. (requirement (F1)). In case the monitoring or processing of this data involves performance requirements (requirement (NF1)), budgets have to be defined in this artifact.

Afterwards, the reaction of the system has to be observed. Hence, a `Causal Connection` between sequences of change and adaptation initiation have to be specified (requirement (F2)). Sensor data on objects that are observed using an in-the-loop mechanisms cannot be predicted but queried whether a certain value is matched. From this, an adaptation decision can be predicted and examined as well. The specification of causal connections can be obtained by defining which symptoms and in which change requests are expected to be produced in a certain context state. To support this inference it can be beneficial to capture the `Environment Structure`. Thus, the effects of past changes can be stored as configuration states during the simulation or generation state an can be taken into account when symptoms are derived. If their are time restrictions on adaptation initiation, budgets have to be specified again (requirement (NF2)).

In the next step, it has to be specified which symptoms end up in which `Adaptation Plans` (requirement (F3)). The respective model maps symptoms to a certain operational set of changes, that are going to be applied against the SAS. In an additional artifact, the system's externally observable behavior has to be modeled in form of a `Service Specification`. This informational artifact captures how the observable interface protocol of the SAS changes in a certain adaptation mode (requirement (F5)). Thus, it involves also propositions on the interaction between the produced adaptation plans and the running processes (requirement (F4)). Finally, the system action's impact on the environment can be examined by predicting these actions in the behavioral system model (cf. requirement (F6)). All qualitative expectations concerning the specified service

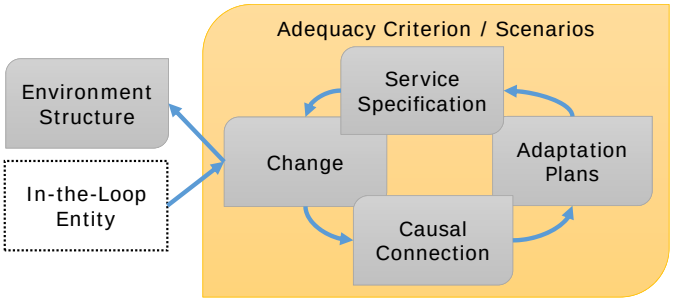


Fig. 9: The Counter Feedback Loop principle.

and its behavior after the adaptation's execution can be taken into account by budget values (requirement (NF3)). As the selection of test sequences through the modeled scenarios is indeterminstic, an adequacy criterion or simulation heuristic has to be specified (requirement (C)).

In summary, the validation model consists of relevant scenarios that stress the system, and oracles that map the scenarios' inputs to the intermediate or final outcomes of each feedback loop component. The different requirements establish a set of concerns that have to be contained in a model and specifiable by its metamodel. By separating these concerns, as proposed by our Counter Feedback Loop principle, different aspects of validation can be decoupled.

VII. Conclusion and Future Work

In this paper, we extended our original work [1] where we applied a customized Failure Mode and Effects Analysis (FMEA) to a conceptional SAS based on the minimal structural assumptions of MAPE-K. We derived a Failure Domain Model in order to provide a system in which faults, errors and failures can be classified. Subsequently, we derived ten distinct failure scenarios that may occur in the process of adaptation. By building a fault dependency graph, we visualized potential cyclic propagation of failures in such systems. In consequence, a set of *founded* modeling requirements were stated that all can be mapped to one or more of the described failure scenarios. Based on these foundations a systematic analysis of SAS is possible comprising failure properties, occurrence, and propagation. A well-designed MBT framework is comprehensive if all presented requirements are fulfilled and the respective assurances are considered.

The extensions of this paper are threefold. Firstly, we exemplified the analysis with our HomeTurtle system in order to clarify the complete process. Secondly, we improved the consideration of non-functional properties that have to be dealt with in validation. Thirdly, we propose to use either Model-based Testing or Simulation for examining SAS against their requirements. Both methods are based on models, whose necessary information have proposed in this paper as well. Based on this premises, test engineers are equipped with indicators when building appropriate generation or simulation frameworks.

For further investigation, it is necessary to instantiate the identified requirements for real-world SAS systems. If implementations can be mapped to several adaptivity frameworks and express the majority of necessary test cases, our approach can be attested substantial and generic.

ACKNOWLEDGMENTS

This research has received funding within the project #100084131 by the European Social Fund (ESF) and the German Federal State of Saxony, by Deutsche Forschungsgemeinschaft (DFG) within Collaborative Research Center 912 (HAEC) and the "Center for Advancing Electronics Dresden" (cfaed) as well as T-Systems Multimedia Solutions.

REFERENCES

[1] G. Püschel, S. Götz, C. Wilke, and U. Aßmann, "Towards systematic model-based testing of self-adaptive software," in ADAPTIVE 2013, The Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications, 2013, pp. 65–70.

[2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1, Jan. 2003, pp. 41–50.

[3] M. Broy, M. V. Cengarle, and E. Geisberger, "Cyber-physical systems: Immanent challenges," in Large-Scale Complex IT Systems. Development, Operation and Management. Springer, 2012, pp. 1–28.

[4] T. M. King, D. Babich, J. Alava, P. J. Clarke, and R. Stevens, "Towards self-testing in autonomic computing systems," in Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems, ser. ISADS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 51–58.

[5] S. S. Kulkarni and K. N. Biyani, "Component-based software engineering." Springer, 2004, ch. Correctness of Component-based Adaptation, pp. 48–58.

[6] B. H. C. Cheng, D. Lemos, H. Giese, P. Inverardi, and J. M. et al., "Software engineering for self-adaptive systems: A research roadmap," in Dagstuhl Seminar 08031 on Software Engineering for Self-Adaptive Systems, 2008.

[7] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, 2004, pp. 11–33.

[8] M. Utting, Practical Model-based Testing: A Tools Approach. Morgan Kaufmann, 2007.

[9] H. E. Roland and B. Moriarty, System safety engineering and management 2nd edn. John Wiley & Sons, Chichester, 1990, ch. Failure mode and effect analysis.

[10] Z. Wang, S. Elbaum, and D. S. Rosenblum, "Automated generation of context-aware tests," 29th International Conference on Software Engineering (ICSE), 2007, pp. 406–415.

[11] V. Dehlen and A. Solberg, "DiVA Methodology (DiVA Deliverable D2.3)," https://sites.google.com/site/divawebsite, visited 02/01/2014, 2010.

[12] A. Maaß, D. Beucho, and A. Solberg, "Adaptation Model and Validation Framework – Final Version (DiVA Deliverable D4.3)," https://sites.google.com/site/divawebsite, visited 02/01/2014, 2010.

[13] F. Munoz and B. Baudry, "Artificial table testing dynamically adaptive systems," arXiv preprint arXiv:0903.0914, 2009.

[14] D. B. Abeywickrama, N. Hoch, and F. Zambonelli, "Simsota: Engineering and simulating feedback loops for self-adaptive systems," in Proceedings of the International C* Conference on Computer Science and Software Engineering, ser. C3S2E '13. ACM, 2013, pp. 67–76.

[15] K. Nehring and P. Niggesmeyer, "Testing the rconfiguration of adaptive systems," in ADAPTIVE 2013, The Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications, 2013, pp. 14–19.

[16] C. Piechnick, S. Richly, S. Götz, C. Wilke, and U. Aßmann, "Using role-based composition to support unanticipated, dynamic adaptation – smart application grids," in Proceedings of ADAPTIVE 2012, The Fourth International Conference on Adaptive and Self-adaptive Systems and Applications, 2012, pp. 93–102.

[17] "MIL-STD-1629A (1980)," Procedures for performing a failure mode, effect and criticality analysis. Department of Defense, USA.

[18] H. Sozer, B. Tekinerdogan, and M. Aksit, Archtitecting Dependable Systems IV. Springer, 2007, ch. Extending Failure Model and Effects Analysis Approach for Reliability Analysis at the Software Architecture Design Devel.

[19] B. Tekinerdogan, H. Sozer, and M. Aksit, "Software architecture reliability analysis using failure scenarios," Journal of Systems and Software, vol. 81 (4), 2008, pp. 558–575.

[20] J. Dugan, Handbook on Software Reliability Engineering. McGraw-Hill, New York, 1996, ch. 15. Software System Analysis Using Fault Trees, pp. 615–659.

[21] S. Hallsteinsen, E. Stav, A. Solberg, and F. J., "Using product line techniques to build adaptive systems," in 10th International Software Product Line Conference, 2006.

[22] B. Morin and A. Solberg, "Reference architecture (DiVA – deliverable D3.3)," https://sites.google.com/site/divawebsite, visited 02/01/2014, 2010.

[23] J. Anderson, R. Lemos, S. Malek, and D. Weyns, "Software engineering for self-adaptive systems," B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Modeling Dimensions of Self-Adaptive Software Systems, pp. 27–47.

[24] G. Blair, N. Bencomo, and R. B. France, "Models@run.time," Computer, vol. 42, no. 10, 2009, pp. 22–27.

[25] G. L. Nemhauser and L. A. Wolsey, Integer and combinatorial optimization. Wiley Interscience, 1999.

Runtime Variability in Online Software Products: A Comparison of Four Patterns

Jaap Kabbedijk, Slinger Jansen, and Thomas Salfischberger
Department of Information and Computing Sciences
Utrecht University, The Netherlands
Princetonplein 5, 3584 CC, Utrecht
J.Kabbedijk@uu.nl, Slinger.Jansen@uu.nl, Tomas@salfischberger.nl

***Abstract*—Business software is increasingly moving towards the cloud. Because of this, variability of software in order to fit requirements of specific customers becomes more complex. This can no longer be done by directly modifying the application for each client, because of the fact that a single application serves multiple customers in the Software-as-a-Service paradigm. A new set of software patterns and approaches are required to design software that supports runtime variability. This paper presents two patterns to solve the problem of dynamically adapting functionality of an online software product; the Component Interceptor Pattern and the Event Distribution Pattern. Additionally, it presents two patterns to dynamically extent the data model; the Datasource Router Pattern and the Custom Property Object Pattern. The patterns originate from case studies of current software systems and are reviewed by domain experts. An evaluation of the patterns is performed in terms of security, performance, scalability, maintainability and implementation effort, leading to the conclusion that the Component Interceptor Pattern and Custom Property Object Pattern are best suited for small projects, making the Event Distribution Pattern and Datasource Router Pattern best for large projects.**

***Keywords*—architectural patterns, quality attributes, software architecture, variability.**

I. INTRODUCTION

This research has previously been published as conference paper [1] and is extended with a pattern description method and presentation and comparison of two dynamic datamodel extension patterns.

Software as a Service (SaaS) is a rapidly growing deployment model with a clear set of advantages to software vendors and their customers. SaaS allows vendors to deploy changes to applications more rapidly, which increases product innovations while reducing support-costs as only a single version is to be supported concurrently [2]. In the SaaS deployment model, a single application serves a large number of customers. These customers are called tenants, which can be a single user or an organisation with hundreds of users. Because all tenants use the same application, the cost of development and setup of the application can be amortized over all contracts.

The multi-tenant deployment model requires the application to be aware of different tenants and their users, for example in separating the data visible to different groups of users. We define multi-tenancy as: "the property of a system where multiple varying customers and their end-users share the system's services, applications, databases, or hardware resources, with the aim of lowering costs". Database designs for multi-tenant aware software require specialized architecture principles to accommodate multiple tenants [3]. One of the challenges in multi-tenant application architectures is the implementation of tenant-specific requirements [4]. Variability of software to fit requirements of specific customers can no longer be done by directly modifying the application for each client or product group, as is customary in Software Product Lines [5]. Because a single application serves multiple customers, only one instance of a product exists, making SPL approaches unusable.

Runtime variability in online software products needs to be enables by a degree of configurability. A new set of software patterns and approaches are required to design software that supports runtime variability. The patterns vary in impact on the technical properties of the software like performance and maintainability, impact on the cost-drivers of the SaaS business model, and the requirements they can fulfil. New patterns are needed for both the data level and instance level of the application. We propose two dynamic functionality adaptation patterns to implement variability at instance level and two dynamic datamodel extension patterns to enable variability at data level. All patterns are evaluated and compared in terms of situational suitability.

The concepts of variability and quality attributes are explained in Section II, after which the expert evaluation used is explained in Section III. Section IV explains how patterns are described in this paper, i.e., functional, system and implementation level. The COMPONENT INTERCEPTOR PATTERN and the EVENT DISTRIBUTION PATTERN, two patterns both solving the problem of dynamically adapting functionality of online business software, are presented in Section V. Section VI presents the DATASOURCE ROUTER PATTERN and CUSTOM PROPERTY OBJECT PATTERN, which introduce variability in the datamodel of online software products. All patterns are compared in terms of security, performance, scalability, maintainability and implementation effort. A concluding overview, presenting the best suitability for all patterns van be found in Section VII.

Please note; in the text we set pattern names in SMALL CAPS according to the convention by Alexander et al. [6].

II. RELATED WORK

Software Patterns - Object oriented design patterns were first introduced by Gamma, Helm, Johnson and Vlissides [7] who define design patterns as recurring patterns of classes

and communicating objects in many object-oriented systems. They state "each design pattern systematically names, explains, and evaluates an important and recurring design in object-oriented systems". We distinguish the patterns described in this research from the original object oriented design patterns by using the name software design patterns. We intend to describe software design patterns for variability techniques in a multi-tenant context in a similar manner to the object oriented design patterns described by Gamma et al. [7].

Others have, based on the first set of design patterns, researched the best methods for describing and communicating design patterns for later reuse. Evits and Hinchcliffe [8], for example, apply UML to design patterns and proposes a modeling technique based on UML-modeling. The same approach is taken by Mapelsden, Hosking and Grundy [9] in their proposal for the Design Pattern Modeling Language (DPML). DPML provides a method for the specification of design patterns as well as a notation linking the elements of design patterns in DPML to UML model elements. They consider three forms, the pattern specification, the pattern instantiation and the final UML object model of the instantiation. In a later publication, Mapelsden et al. present tool-support for the DPML to automatically transform a pattern specification into a pattern instantiation and to maintain consistency between pattern specification, pattern instantiation and the UML object model [10].

[11] discuss the need of a more formal design pattern description language to support Computer Aided Software Engineering (CASE) tools. They describe previous pattern description languages based on generic UML diagrams annotated with natural language constraints as a problem for CASE tools. However, their main concern is the fact that previous pattern description approaches tend to describe a single implementation of the pattern where the true meaning of the pattern is lost to a description of implementation details. The running example is the Abstract Factory Pattern as described by Gamma et al. [7]. The proposed solution is to apply three separate layers of modeling, the role-model, type-model and class-model. At the highest level of modeling the role-model only describes the parts of a design pattern and their relative roles and interaction. The type-model is a refinement of the role-model where details like implemented methods are added. The type-model should according to Lauder and Kent [11] be supplemented by a textual description of the motivation, trade-offs and known uses. The final refinement of the type-model is the class-model, where a concrete implementation is described as is the case in previous pattern description languages.

Variability - The field of software variability has been the subject of research from both the modeling perspective as well as the technical perspective [12]. The application of variability modeling as used in product line variability [13] to software as a service environments has been described by Mietzner, Unger, Titze and Leymann [14]. Variability modeling as discussed in the aforementioned works contributes to the understanding of where the application architecture needs to be able to accommodate change or extension. Patterns play an important role in modeling and solving variability in software products [15].

Svahnberg, van Gurp and Bosch [16] propose feature diagrams as a modeling technique to describe the different variants of feature in a software product. They use their feature diagrams as the basis for a method to identify variability in a product, constrain this variability, pick a method of implementation for the variability and further manage this variability point in the application lifecycle. The main difference from the objectives of our research is that Svahnberg et al. describe implementation techniques for variability per installation instance of the software, whereas we focus on *runtime* variability in a multi-tenant context.

Quality Attributes - Benlian and Hess [17] identify *security* as one of the most important risk-factors perceived, followed by performance risks. To assess security risks, SaaS vendors need to include security as a quality attribute in their design of the architecture. This leads to security as the first desired quality attribute for business SaaS. *Performance* as an important factor to SaaS users is closely related to the most important factor, i.e., cost [17]. When performance is insufficient, clients are lost, when the system uses too many resources to gain an acceptable level of performance, cost is increased. A SaaS vendor must thus assess the possible performance impact of changes to the software. To control cost in business SaaS, the SaaS vendor needs to utilize its opportunities for scalability to decrease the cost of hardware or hosting fees (e.g., using scalable software to make optimal use of cloud-hosting).

Another cost driver in SaaS is the *cost of development* and *maintenance* of the software product. Maintenance cost is generally decreased by having to maintain only a single version instead of multiple previous releases. On the other hand, this maintainability cost-saving must not be lost while implementing runtime variability. Thus, scalability and maintainability are also desired quality attributes for business SaaS. Another way the implementation of runtime variability will influence product cost is through implementation-cost. Development is a cost-driver for SaaS, thus if one or more specialized developers are required to implement a certain pattern this will influence the final product cost.

The identified quality attributes are the following:
Security - The ability to isolate tenants from each other and the possible impact of security breaches in custom components on other parts of the system.
Performance - The utilization of computing, storage and network resources by the application at a certain level of usage by clients.
Scalability - The relative increase in capacity achieved by the addition of computing, storage and network resources to the system as well as the flexibility with which these resources could be added to the system.
Maintainability - The ease with which the system can be extended and potential problems can be solved.
Implementation Effort - The effort required to implement and deploy a specific system.

III. Research Approach

In order to gather the patterns in this research, a design science approach [18] was used in which the initial solutions are observed in case studies in which one of the authors took part as a consultant. The solutions are implemented in current commercial software products. The architecture description

and source code off the software products is examined and checked if runtime variability in functionality or datamodel is supported. Whenever this is the case, the solution used is documented and the consequences of the solution analyzed. Solutions observed in at least three products are presented as patterns and are evaluated by two domain experts to ensure correctness and usefulness. The evaluation of the cases by experts enhances the validity of the cases [19]. During each evaluation session, patterns are discussed with an expert, in a semi-structured way. Standard questions related to the quality attributes are asked, after which issues are freely discussed per quality attribute.

The first expert selected is a senior software architect in an international software consulting firm specialized in large scale development of Enterprise Java applications. His role is to investigate technologies and methodologies to help design better architectures resulting in faster development and more extensible software. A recent project includes a multi-tenant administrative application storing security sensitive data for multiple organizations. The second expert is a technology director and lead architect for an application used in distributed statistics processing of marketing data, previously working in software performance consulting for web-scale systems. His experience lies in the field of high-performance distributed computing. The application his company works on focuses of low-latency coordinated processing of large volumes of data to calculate metrics used for marketing. Performance and scalability are important areas of expertise for their product.

IV. Pattern Description Method

The use of patterns in order to describe multi-tenant systems is different from the way object oriented design pattern are commonly applied. An object oriented design pattern describes common solutions to problems in object oriented software design. The most important difference between object oriented software design and the design of multi-tenant systems is that the problem scope in multi-tenant systems is not limited to only the objects in object oriented software. The software system is considered not only to be a set of source files, but to include supporting systems like databases, message-bus and infrastructure.

The needs for a description language for the discussed design patterns thus includes the need to describe any necessary

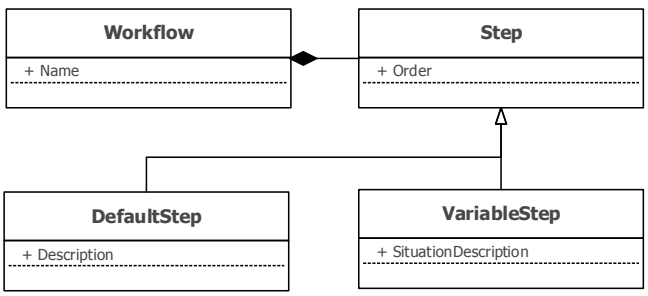


Fig. 1: Example UML class diagram

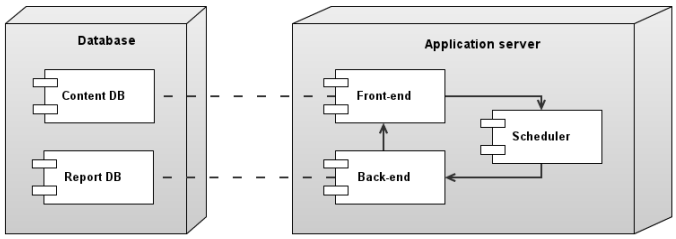


Fig. 2: Example UML Deployment Diagram

characteristics of the supporting systems and auxiliary materials. When considering design patterns for software systems we propose a combination of description techniques at different levels similar to Lauder and Kent [11]. Instead of modelling different levels of detail and abstraction within only object oriented design, different levels of the software architecture including supporting systems have to be modelled. The levels we propose to describe online systems are:

1) Functional level
2) System level
3) Implementation level

Functional level - This level describes the functional intention of the pattern in a technical context. Multiple different patterns can share the same model at functional level, because several patterns can be designed to reach the same functional effect with, for example, different performance and scalability characteristics. For the graphical modelling of the functional level, UML class diagrams are used as shown in Figure 1. This diagram captures the functional situation resulting from application of the pattern without considering implementation of pattern instantiation details.

System level - This level models the overview of the software including supporting systems after the application of the pattern. Interaction among different components within and between systems as a result of the implemented pattern are

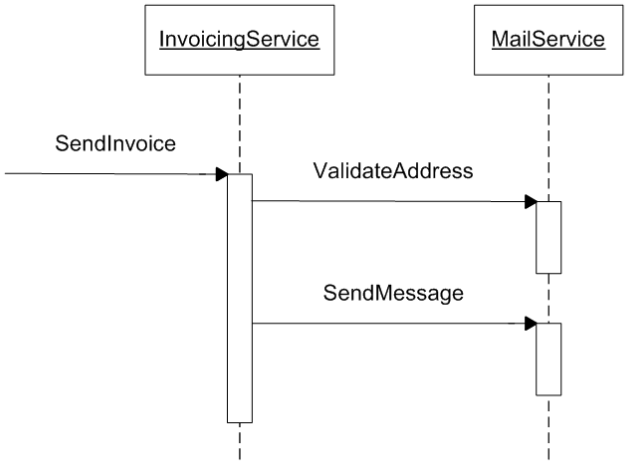


Fig. 3: Example Sequence Diagram

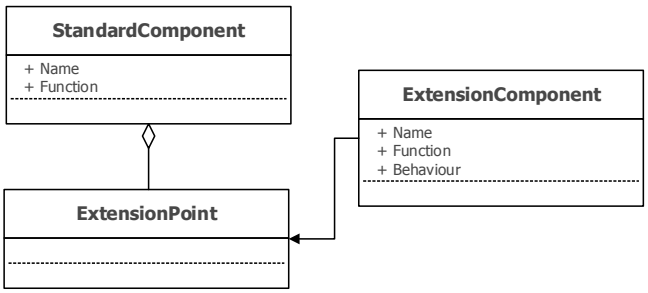


Fig. 4: Functional Model for adapting functionality

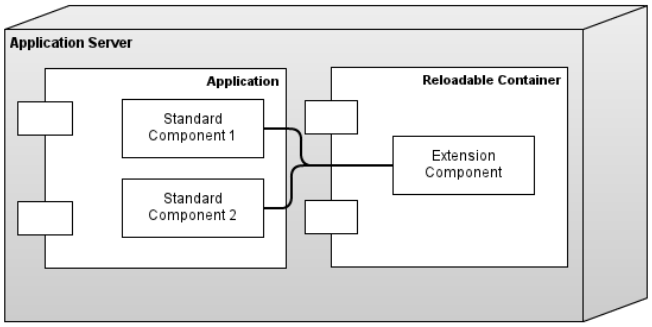


Fig. 5: Component Interceptor Pattern: System Model

shown. A UML deployment diagrams [20] is used to describe this level (see Figure 2 for an example).

Implementation level - The third level describes the potential implementation of the pattern. These diagram depicts a specific implementation of the components of the pattern. The implementation diagram is closely related to the system model, but depicts the method of application of the components in the system model on a more detailed level. Within this research we use a sequence diagram as shown in Figure 3 to illustrate the implementation. This description level should be regarded as a possible way to implement the pattern, but it does not prescribe a specific implementation.

V. Dynamic Functionality Adaptation Patterns

A. Problem Statement

Software product vendors not only need to offer a *data model* that fits an organisation's requirements, *software functionality* also has to meet an organisation's processes [21]. When tailor-made software is developed, it is possible to set the requirements to exactly match the processes of a specific organisation. For standard online software products this is not possible and differences between requirements of organisation have to be addressed at runtime.

A requirement for the ERP system of a manufacturing company could be to send a notification to the department responsible for transportation if tomorrow's batch will be larger than a certain size. If this requirement is not met by the software product selected, the company could either decide to select another software product or develop a tailor-made application that does meet their requirements.

To allow for the addition of extra functionality in the application, a solution that allows to configure this functionality is needed. This functional situation is modeled in Figure 4, the envisioned functional situation. The *StandardComponent* is a normal component of the software with default functionality, this component has a set of *ExtensionPoints*. An *ExtensionPoint* is a location within the normal workflow where there is a possibility to add or change functionality. This functionality is specified in an *ExtensionComponent*, which contains the actual functionality that is to be executed at the specified *ExtensionPoint*.

Two different patterns are identified, both offering a solution to dynamically adding functionality to a software product.

B. Component Interceptor Pattern

The Component Interceptor Pattern as depicted in Figure 5 consists of only a single application server. Interceptors are tightly integrated with the application, because they run in-line with normal application code. Before the *StandardComponent* is called the interceptors are allowed to inspect and possibly modify the set of arguments and data passed to the standard component. To do this the interceptor has to be able to access all arguments, modify them or pass them along in the original form. Running interceptors outside of the application requires marshalling of the arguments and data to a format suitable for transport, then unmarshalling by the interceptor component and again marshalling the possibly modified arguments to be passed on to the standard component that was being intercepted. This is impractical and involves a performance penalty [22].

Running the extension components inside the application-server while supporting runtime variability requires support for adding and changing interceptors at runtime. The system model depicts this requirement in the form of a reloadable container. In some implementations this could be as simple as changing a source file, because the programming platform used will interpret source code on the fly. Other platforms require special provisions for reloading code, such as OSGi for the Java platform or Managed Extensibility Framework for the .NET platform.

Figure 6 depicts the interaction with interceptors involved. Interaction with standard components that can be extended goes through the interceptor registry. This registry is needed

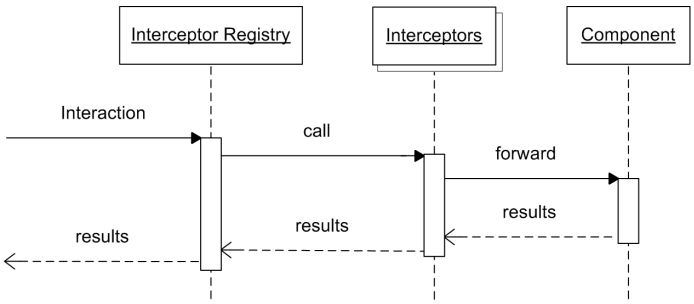


Fig. 6: Component Interceptor Pattern: Sequence Diagram

to keep track of all interceptors that are interested in each interaction. Without the registry the calling code would have to be aware of all possible interceptors. As depicted, multiple interceptors can be active per component. It is up to the interceptor registry to determine the order in which interceptors will be called. An example strategy would be to call the first registered interceptor first or to register an explicit order when registering the interceptors.

Each interceptor has the ability to change the data that is passed to the standard component, modify the result returned by the standard component, execute actions before or after passing on the call or even skip the invocation of the next step all together and immediately return. Immediately returning would for example be used when the interceptor implements certain extra validation steps and refuses the request based on the outcome of the validation. As a result of these possibilities the interceptors must be invoked in-line with the standard component, the application cannot continue until all interceptors have finished executing.

C. Event Distribution Pattern

In the event distribution pattern the application generates events at extension points, which are distributed by a broker. At each extension point the standard component is programmed to send an event indicating the point and appropriate contextual data (e.g., which record is being edited) to a broker. For example in a CRM system the standard component for editing client-records sends a *ClientUpdated* event with the ID of the client that was edited. Extension components listen for these events and take appropriate actions based on the events received. In the example of a *ClientUpdated* event, an extension component could be developed that sends a notification to an external system to update the client details there.

The system model in Figure 7 depicts the distributed nature of the EVENT DISTRIBUTION PATTERN. Standard components run in the application server, sending events to a central broker, which can be run outside of the application. Extension components are isolated and can be on a separate physical server or run as separate processes on the same server depending on capacity and scale of the application. Components are loosely coupled, sharing only the predefined set of events. The standard components are unaware of which extension components listen for their events, execution of extension components is decoupled from the standard components. Executing the extension components separately allows for independent scalability of these components. Depending on system load and the volume of events each component listens for, it is possible to allocate the appropriate amount of resources to each component. Because there is no interaction between listeners, it is possible to execute all listeners in parallel if appropriate for the execution environment.

Standard components publish events to the broker as depicted in the sequence diagram in Figure 8. The activation of the standard component not necessarily overlaps with its listeners. After publishing the event, a standard component is free to continue execution. Depending on the fault tolerance and nature of the events it is up to the standard component to make a trade-off between guaranteed delivery at a higher latency by waiting on the broker system to acknowledge reception of the event or continue without waiting for such an acknowledgement. If, for example, an event is only meant to prime a cache for extra performance the loss of such a message would not impact critical functionality of the system while waiting for the message might mitigate any performance gains. On the other hand, if an event is used for updating an external system for which no other synchronization method is available the system needs guaranteed delivery to function correctly. At design time this decision can be made on an event by event basis depending on the capabilities of the messaging system used.

Because of the one-way nature of events and decoupled execution of extension components it is not possible for an *ExtensionComponent* to stop standard functionality from happening. In the observed system this was solved by allowing *ExtensionComponents* to execute a compensating action in their listener. The compensating action is sent from the listener component back to the system independently of the original action that caused the event. An example of such a compensating action is an extension component that monitors changes to certain records and reverts the change in case special conditions are met. This approach has the added benefit that any changes made by extension components are clearly visible in audit logs, which simplifies tracing possibly unexpected system behaviour back to an *ExtensionComponent*.

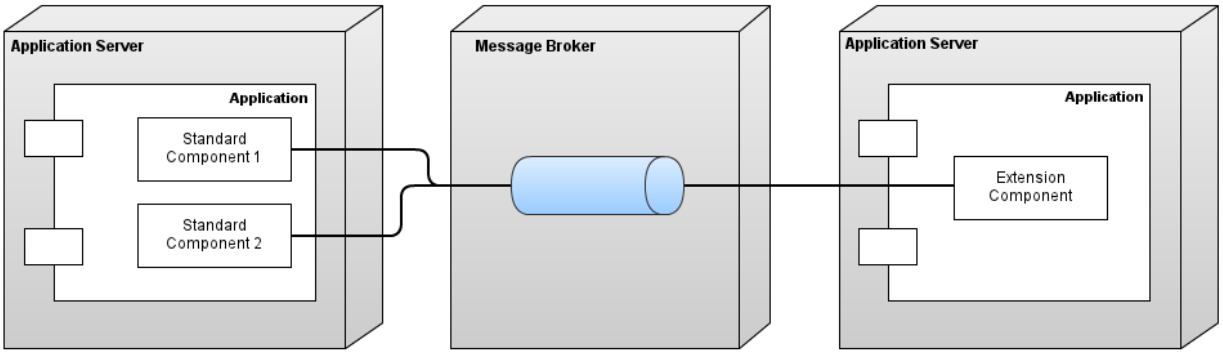


Fig. 7: Event Distribution Pattern: System Model

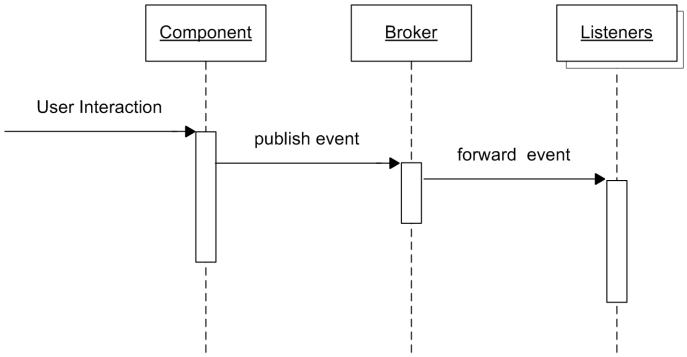


Fig. 8: Event Distribution Pattern: Sequence Diagram

D. Pattern Comparison

This section presents an analysis of both patterns on the five presented quality attributes.

1) Security: When adapting functionality of an application, there is always the possibility of introducing new security vulnerabilities. This is an inherent risk of extending an application. The variability patterns do, however, influence how much larger the attack surface becomes and how well a breach in one of the components is isolated from other components. In the COMPONENT INTERCEPTOR PATTERN, the code handling the new functionality becomes part of the application and will have the ability to execute arbitrary code within the context of the main application as depicted in Figure 5. It will also have full access to any parameters passed to intercepted functions as well as any returned values. A security breach in the extension components (interceptors) is not isolated to only those components, unless extra security measures are implemented to separate the components from the main application. Adding extra security measures, however, does have an impact on the performance efficiency of the application.

The EVENT DISTRIBUTION PATTERN isolates the extension components from the application by executing them in a separate context based on incoming events as depicted in Figure 6. This execution in a separate context allows for more isolation between extension components and the main application components. The components also have far more limited access to standard functionality, because any change the component wants to make has to go through explicitly exported APIs or messages. Combined with event-sourcing, any change to data as a result of custom functionality is fully traceable including the original values [23].

2) Performance: The COMPONENT INTERCEPTOR PATTERN executes interceptors within the context of the application. This results in little overhead when executing the extension components, because data does not need to be marshalled, unmarshalled and transferred between applications. However, for security reasons it could be necessary to separate the interceptors from the main application as described in the previous section. This removes one of the performance advantages of the component interceptor pattern because data must be transferred between the different contexts.

Applications implementing the EVENT DISTRIBUTION PATTERN require the setup of a message broker that handles all events coming from the application and going into the extension components. This requires extra processing and network resources and in the case of durable message delivery mechanisms also storage resources reading and writing the messages. To transfer the events from the application via a message broker to the extension components the events must be marshalled into a format suitable for transferring over a network and unmarshalled upon reception by the extension component, these steps add non-trivial cost to the operations.

3) Scalability: Applications using the COMPONENT INTERCEPTOR PATTERN will execute interceptors within the context of the application. This has performance advantages described in the previous section, however, the interceptors cannot be scaled independently of the application. When a high number of interceptors exists requiring significant resources the application as a whole needs more application servers to execute. The interceptors must be available to all application servers in that case.

On the other hand, the EVENT DISTRIBUTION PATTERN decouples the execution of the event handlers from the application by running them on a logically separate application server. Because events are handled outside the execution flow of the standard components they can also be distributed to multiple systems. Adding extra application servers subscribing to the same events in the message broker the processing capacity of events could increase linearly. For the EVENT DISTRIBUTION PATTERN this requires a message broker system that is able to handle the increasing numbers of messages. Those systems are available off the shelf from open source projects like Fuse Message Broker, JBoss Messaging, RabbitMQ and commercial offerings like Microsoft BizTalk, Oracle Message Broker or Cloverleaf.

4) Maintainability: When adapting the functionality of an application, maintainability is also affected by the necessity to make sure future extensions and modifications are compatible with any custom functionality implemented for tenants. This is a trade-off between the flexibility and depth with which *ExtensionComponents* can affect the application and the impact that changes to the application will have on the *ExtensionComponents*. As an example of the aforementioned trade-off, a simple system with only a single *ExtensionPoint* will have a much lower impact on maintainability than a complex system with a very high number of *ExtensionPoints*. This however affects both patterns equally.

The way the patterns decouple *ExtensionComponents* from *StandardComponents* is however a differentiating factor. In the COMPONENT INTERCEPTOR PATTERN the *ExtensionComponent* is more tightly integrated with the *StandardComponent* because calls to a *StandardComponent* at an *ExtensionPoint* go through the interceptor providing all parameters and return values of the call. When changing calls by adding or removing parameters this will directly affect the input of each *ExtensionComponent* registered from that *ExtensionPoint*. When applying the event distribution pattern the integration is more decoupled because calls to StandardComponents are not directly affected by the *ExtensionComponents*. Instead the *ExtensionComponent* receives a standardized event-message and uses a provided API to send any changes or other actions back to the application. This allows for changes to

the *StandardComponent* without changing the event-messages going to the *ExtensionComponent*. At the same time the API used by *ExtensionComponents* to influence the application can be kept stable for small changes or versioned to support future compatibility using methods like the one described by Weinreich, Ziebermayr, and Draheim [24].

5) Implementation Effort: When implementing a pattern for adding functionality to an application we distinguish two factors determining the implementation effort. The first factor is the direct effort required to implement the pattern in the system, e.g., adding *ExtensionPoints* to the *StandardComponents* of the application. The second factor is the effort necessary to implement *ExtensionComponents*. Later changes to the components might also require development effort, this is however excluded from implementation effort because it is covered under maintainability. Both patterns require the definition and implementation of *ExtensionPoints*, the way these points are implemented differs per pattern. When implementing the COMPONENT INTERCEPTOR PATTERN it is necessary to setup an Interceptor Registry and modify calls to *StandardComponents* to go through the Interceptor Registry.

In the EVENT DISTRIBUTION PATTERN, a message broker system must be setup to handle the event-messages flowing from *StandardComponents* to *ExtensionComponents*. The application still has to be modified at the ExtensionPoints to send the event-messages belonging to that *ExtensionPoint*. A larger difference between the two patterns emerges in the way they influence the system. Using component interceptor pattern each interceptor has full access to the application because it executes within the same context. Communication with *StandardComponents* from within *ExtensionComponents* could use normal function-calls just like any other part of the system. This differs from the event distribution pattern where the *ExtensionComponents* execute in a separate environment outside the context of the *StandardComponents*. Any interaction between *ExtensionComponents* and *StandardComponents* needs to go through an external interface. Depending on the type of system and the requirements for interaction this requires the development of some sort of (webservice-)API for the *ExtensionComponents* to use.

The second factor of implementation effort, the effort required to implement ExtensionComponents, affects both patterns. In the COMPONENT INTERCEPTOR PATTERN the implementation requires the development of an interceptor, which executes the correct behaviour when certain conditions are met. The EVENT DISTRIBUTION PATTERN requires the development of ExtensionComponents, which listen for the right messages and execute the correct functionality when certain conditions are met.

Please see Table I for an overview of the evaluation of both patterns. Plus and minus signs are used to indicate whether a characteristic is positive or negative. Keep in mind all scores are relative scores compared to the other pattern.

VI. DYNAMIC DATA MODEL EXTENSION PATTERNS

A. Problem Statement

Organisations within the same or different market all strive to differentiate themselves, which results in numerous different working processes each with specific requirements for the supporting software systems. Additionally, across markets and jurisdictions differences exist in regulations and standards which require the storage and reporting of different data for each organisation. Organisations will thus set varying requirements to store data specific to their needs. These requirements could be met by software specifically designed for the market in which this organisation operates or even software tailored to the needs of one specific organisation. Specializing software of a small market or even single organisation decreases the number of possible clients for the software vendor and increases the cost per client. A software product that provides enough variability on the data model to meet organisation specific requirements will decrease cost and attract clients that cannot currently be serviced by software products unable to meet their specific requirements. Extension of the data model by creating additional fields to store data that are specific to an organisation or their working processes is a common requirement [25].

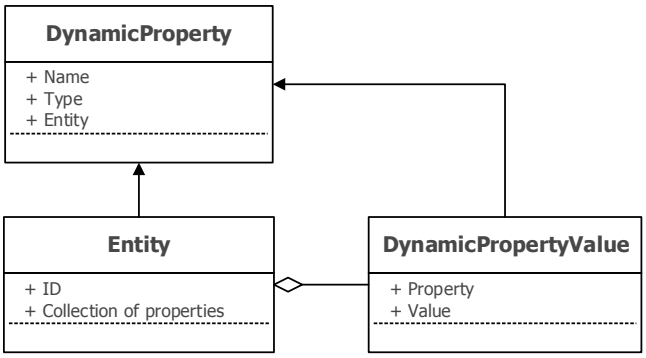


Fig. 9: Functional Model for datamodel extension

In case of standardized software, where this requirement is not met by the default installation of the software, an extension of the existing data model is required. Figure 9 depicts the envisioned functional situation, storing custom properties of entities in the domain model. The depicted *Entity* is the original entity in the application domain model which contains a *DynamicPropertyValue* and has a relation to a *DynamicProperty*. This property is configured for a specific tenant and holds settings like for example a name and expected data-type.

B. Datasource Router Pattern

In this pattern, the application uses a different database instance (or schema) for each tenant. Custom properties are then added to the database as normal fields. Each component in the application accesses this database through the *Datasource Router*. The *Datasource Router* component determines which database is to be used (based on the tenant the current user belongs to) and routes all access to the right database automatically. The other components can thus work without being aware of the fact that the application is actually serving multiple tenants using different databases.

The system model, which is shown in Figure 10, describes the overview of the system when implementing the DATASOURCE ROUTER PATTERN. As shown, the application uses

TABLE I: OVERVIEW OF BOTH DYNAMIC FUNCTIONALITY ADAPTION PATTERNS

	Component Interceptor Pattern	Event Distribution Pattern
Security	- Extension components execute within application scope	+ Isolation of extension components and full traceability of actions by extension components
Performance	+ Direct execution of extension components	- Network overhead for calling extension components - The broker system requires extra resources
Scalability	- No independent scaling of extension components - Does not scale to high number of extension components	+ Independent scaling of extension components + Extension components cannot delay standard components - Requires scalable message-broker system
Maintainability	- Tight coupling of extension components	+ Loose coupling of extension components
Implementation Effort	+ Direct communication with standard components + Access to all data by design.	- Requires the setup of a message broker system - Requires a separate mechanism to communicate with the application

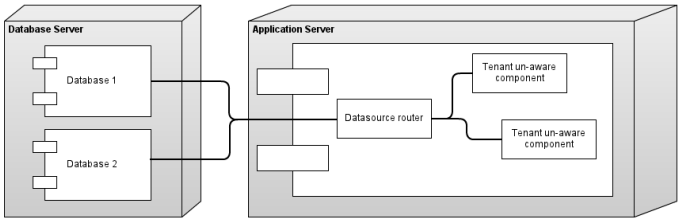


Fig. 10: Datasource Router Pattern: System Model

multiple separate databases (i.e., Database 1 and Database 2 in the figure) to store data for different tenants. Each component accesses the database through a *Datasource Router*, which determines to which database the queries are sent. Due to this isolation the components that access the database never encounter data for multiple tenants at once, since a query will always return results for one and only one tenant, because it is sent to a database that contains only data for a single tenant. This means the components do not need to be multi-tenancy aware in querying the data.

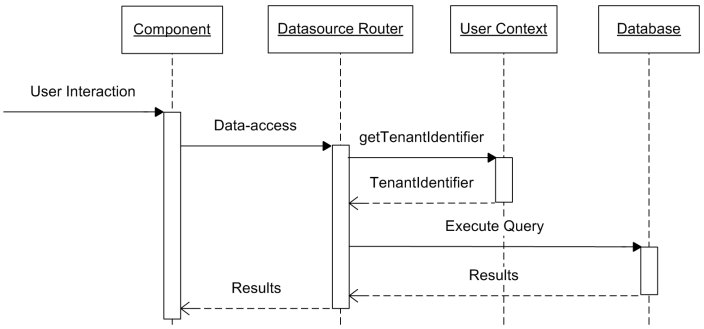


Fig. 11: Datasource Router Pattern: Sequence Diagram

The interaction between tenant-unaware components and the database goes through the *Datasource Router*. The sequence diagram in Figure 11 depicts the interaction from component through *Datasource Router* to the actual database. First, the user interacts with a component, this component requires access to data, which is done through the *Datasource Router*. The *Datasource Router* is then responsible for determining which tenant the current user belongs to, this responsibility is delegated to the *User Context*. It is implementation dependant how this *User Context* is implemented, the only requirement is that it is able to tell the *Datasource Router*, which tenant is to be used in the context of the current request. After determining which tenant is active the *Datasource Router* executes the query on the right database (selected based on the active tenant), the results are then returned to the component, which originally needed access to the data. In this sequence, it is clear that from the perspective of a component requesting data it does not matter how multi-tenancy is implemented in deeper layers. The component is isolated from these choices and the possible complexity involved in selecting the right datasource to use for the current user.

C. Custom Property Object Pattern

When implementing the CUSTOM PROPERTY OBJECT PATTERN, data from all tenants is stored in a single database with a single schema. Any additional data like custom properties is modeled in the design of the application as separate custom property objects, which are stored in the existing static schema. Because all data is stored in a single database components using that data need to be aware of multi-tenancy and explicitly query for data of a specific tenant.

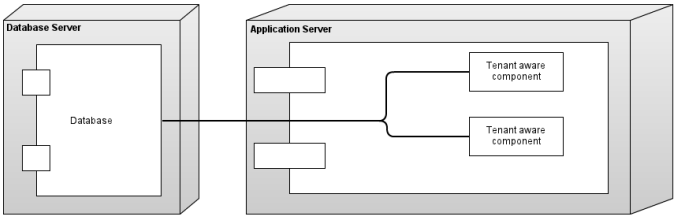


Fig. 12: Custom Property Object Pattern: System Model

This pattern prescribes the storage of all data in a single database, which is accessed by components that are aware of how to filter data for each tenant. In the system model, as depicted in Figure 13, components are aware of multi-tenancy and directly access a single database to query for the data necessary to complete requests. When querying the data it is the responsibility of each component to only query data related to the requested tenant or filter data while processing, to get results only for the current tenant.

As a result of using a single database for all tenants, the other components need to be aware of the context in which

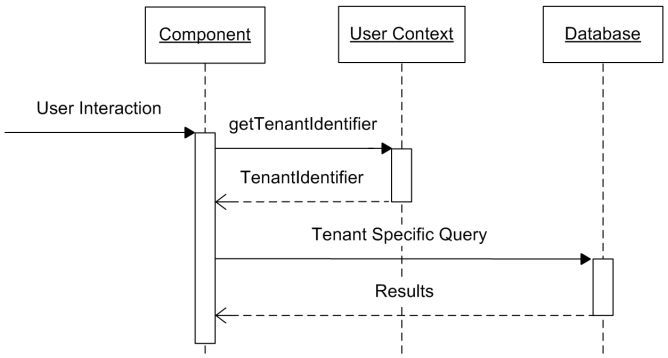


Fig. 13: Custom Property Object Pattern:Sequence Diagram

they operate. When retrieving data the components need to filter the results to only show data for the current tenant. The resulting interaction from component to database is depicted in Figure 13. The component first determines which tenant is currently active, this is done by using the User Context. It is implementation dependant how this User Context determines this, the only requirement is that it is able to tell a component which tenant is to be used in the context of the current request. The component then generates a query that is specific to the current tenant and sends this to the database. It is the responsibility of the component to ensure that the generated query only accesses data for the current tenant and to avoid retrieving data outside of tenant boundaries.

D. Pattern Comparison

1) Security: Comparing the different data storage structures of the DATASOURCE ROUTER PATTERN and the CUSTOM PROPERTY OBJECT PATTERN shows that the DATASOURCE ROUTER PATTERN separates data from each tenant in a separate schema or database. This separation also guarantees that when a query is executed it will only return data for a single tenant without extra efforts from the developer. Because the *datasource router* component is the only component involved in selecting the datasource for a query, the changes of accidentally mixing data from multiple tenants due to programming errors are low. Failing to select a datasource would simply crash the application instead of mixing data from other tenants.

On the other hand, the CUSTOM PROPERTY OBJECTS PATTERN relies on the developers to write queries to only return data from the appropriate tenant. When no precautions are taken in the development and testing process the possibility of accidentally mixing data from multiple tenants is higher than when the DATASOURCE ROUTER PATTERN is used. When a correct filter is not applied in this pattern, users will receiving data from other tenants that should never be visible to them. When implementing this pattern it is critical to implement a strong test and quality assurance system as well as methods for automatically detecting queries that fail to filter data correctly.

At the system level the DATASOURCE ROUTER PATTERN requires a separate database or schema per tenant, these separate instances must all be monitored, updated and secured separately. Automation of security related system administration tasks is important, to ensure that all instances are always in the required state. Failing to implement proper procedures might result in tenant instances being in different states of updates and security related configuration settings. Security procedures for the custom property objects pattern can be simpler, because only a single database needs to be monitored and secured. This single database system is however a more high value target from a security perspective because data from all tenants is stored in a single place.

2) Performance: The CUSTOM PROPERTY OBJECTS PATTERN uses only a single large database or schema, which allows the database server to allocate all resources to one entity. The DATASOURCE ROUTER PATTERN requires a separate database or schema for each tenant, which, depending on the database system used, can result in partitioning of available resources like memory and caches and requiring more network resources to connect to all databases separately. Query efficiency in the custom property objects pattern is dependent upon the design of the database schema.

If the schema is generic, storing all data in field types without type information, the database engine will not be able to apply optimizations for specific datatypes. For example, storing fixed length integers in a variable length BLOB-field does not allow the database engine to make use of the known length of the field for faster searching through the storage structures. Designing the schema to partition data by tenant allows the database to limit the amount of data that is necessary to retrieve when executing a query for a single tenant. This limitation comes naturally for the DATASOURCE ROUTER PATTERN, because the data for each tenant is stored separately.

3) Scalability: Two types of scalability exist; vertical scalability and horizontal scalability. In vertical scalability we consider the amount of added capacity available when increasing the resources of a single system, e.g., adding more memory, more storage or more processing power to a single server. This is naturally limited by the available hardware options and associated costs of those components. Horizontal scalability concerns the scalability of adding more instances instead of increasing capacity in a single system. Horizontal scalability does not have the implied limits of available hardware that exist in vertical scalability, however, achieving perfect horizontal scalability has several challenges in coordination of nodes in a system. In practice this coordination costs resources, which makes it hard to achieve linear scalability in systems that require coordination of their workload.

By applying the CUSTOM PROPERTY OBJECTS PATTERN the application will only use a single database system. This impacts scalability in the application that requires a database system that is able to scale by itself to achieve scalability of the system as a whole. For example, a database system that supports clustering is appropriate to support scalability of the custom property objects pattern. In the DATASOURCE ROUTER PATTERN adding additional sources by moving part of the databases to separate servers is possible and does not require a database system capable of clustering.

The DATASOURCE ROUTER PATTERN is easier to scale out when the amount of tenants increases. An example case is a system currently using two database systems. In this example system, new tenants subscribe to the service and the

TABLE II: OVERVIEW OF BOTH DYNAMIC DATAMODEL EXTENSION PATTERNS

	Datasource Router Pattern	Custom Property Object Pattern
Security	+ Natural separation of datasets + Single point of selecting correct datasource - More datasources to secure and maintain	+ Only a single datasource to secure and maintain - Risk of losing data separation with programming errors
Performance	+ Correct data-types allow for optimizations - Resource partitioning across separate schemas	+ Full resource utilization across all schemas - Loss of optimizations due to lack of type information
Scalability	+ Natural scalability due to separate schemas + No need for scalability support in database	- No inherent scalability in pattern structure - Requires database system capable of scaling
Maintainability	- Large number of possible database schemas must be tested - Problem solving requires schema variants to be included	+ Single static database schema + Custom properties can be handled with generic shared code
Implementation Effort	+ Central component to handle all data-access - Custom properties must be handled in all components	- Requires adaption of data-access in all components - Custom properties must be handled in all components

capacity becomes insufficient to service all tenants. Horizontal scalability is possible by adding two more database systems, effectively doubling the database capacity by allowing the data for new tenants to be stored on the new systems. There is virtually no overhead involved in this addition, because no extra coordination is required between the database systems servicing data for separate tenants.

The CUSTOM PROPERTY OBJECTS PATTERN requires a database system that is able to store all data for all tenants. The database system must in that case support vertical scalability by increasing the capacity of a single system instead of horizontal scalability. The application of a database system that provides a scalability capability is necessary for large deployments of this pattern. The results are dependant upon the effectiveness with which the database system deals with scalability challenges.

4) Maintainability: When extending the application with new functionality both patterns require that the new functionality is aware of any customized objects. For the DATASOURCE ROUTER PATTERN this involves creating a solution able of determining all database schema variations and correctly copying these values. The code involved can be complex because of the need to support various database modifications supported by the underlying database system. In the CUSTOM PROPERTY OBJECTS PATTERN, the extra properties are stored as predefined database objects, which can be handled the same as any other object stored in the database of the application. This means the code to handle the custom properties can be much simpler. A generic system could always handle the custom properties in the same way agnostic of their contents because they are abstracted as normal database objects. For problem solving a similar difference exists.

A problem affecting a single tenant in an application using the DATASOURCE ROUTER PATTERN can be harder to reproduce because of the various schema changes that could be done to the schema for that specific tenant. Because the changes, it is harder to isolate the root-cause of the problem. The CUSTOM PROPERTY OBJECTS PATTERN deals with a fully standardized database schema where the possible types of custom properties are explicitly visible in the design of the system. Because of this it is easier to create correct test-cases for the CUSTOM PROPERTY OBJECTS PATTERN, whereas the DATASOURCE ROUTER PATTERN has much more potential schema-variations, which must be explicitly handled correctly and tested.

5) Implementation Effort: For the DATASOURCE ROUTER PATTERN the initial implementation requires the development of the router component as well as systems to manage and automatically deploy new database instances for new tenants. The other components can however be left unchanged because awareness of the multi-tenant environment is not required. Using the CUSTOM PROPERTY OBJECTS PATTERN, on the other hand, does not require the development of new components or management systems. For this pattern the existing components need to be adapted to query the right data and use appropriate filtering methods. Both patterns require the implementation of code handling the existence of custom properties for entities in the applications data model. This is equal for both patterns and thus of no influence in a comparison on implementation effort.

VII. CONCLUSION

Within this paper two problem domains related to implementing runtime variability in online business software are discussed. Also a pattern description method is proposed, suggestion the use of the following description levels: 1) Functional level, 2) System level and 3) Implementation level.

First, two dynamic functionality adaptation patterns, which are the COMPONENT INTERCEPTOR PATTERN and the EVENT DISTRIBUTION PATTERN are compared in terms of security, performance, scalability, maintainability and implementation effort. Both patterns offer a solution for dynamically adapting functionality of an online software product, but do so in different ways. The COMPONENT INTERCEPTOR PATTERN performs less in terms of *scalability*, because the interceptors can not scale independently of the application. When scaling up in terms of number of servers, the interceptors need to be available to all servers. Related to this issue, the *maintainability* of the COMPONENT INTERCEPTOR PATTERN is also less than that of the EVENT DISTRIBUTION PATTERN. This is caused by the fact the interceptors can not be decoupled from the rest of the system, creating a software product that will be difficult to maintain. The EVENT DISTRIBUTION PATTERN offers more isolation in terms of *security* than the other pattern, but requires more processing and network resources in terms of *performance*. Related to *implementation effort*, the COMPONENT INTERCEPTOR PATTERN is easier to implement, because no message broker or related services are required.

In general, the COMPONENT INTERCEPTOR PATTERN serves best for adapting functionality of small projects, where the EVENT DISTRIBUTION PATTERN is better for large projects, considering the quality attributes described in this paper.

Second, two dynamic data model extension patterns, being the DATASOURCE ROUTER PATTERN and CUSTOM PROPERTY OBJECT PATTERN are presented and evaluated. We conclude that the DATASOURCE ROUTER PATTERN has advantages on *security* by naturally isolating the data for all tenants, *scalability* by allowing for the distribution of tenants across datasources and *implementation* by not requiring all queries and components to be adapted but providing a single router component instead. The custom property objects pattern holds an advantage on performance by allowing better resource utilization, however, extra care is necessary to design an appropriate database schema. The CUSTOM PROPERTY OBJECTS PATTERN also scores better on *maintainability* by allowing standardized handling of the dynamic properties and using a static data model avoiding the need to test every possible variation when adapting the software.

For future work we are currently setting up larger evaluation sessions in which different patterns will be evaluated using experts. The evaluation of patterns is particularly difficult, because you should evaluate an abstract solution instead of a specific implementation. We are working on a structured method for comparing sets of patterns and making use of the implicit knowledge of experts. By doing this, we aim at evaluating the *solution*, instead of just an *implementation*.

ACKNOWLEDGMENT

The authors would like to thank Allard Buijze and Koen Bos for helping in reviewing the results of the research.

REFERENCES

[1] J. Kabbedijk, T. Salfischberger, and S. Jansen, "Comparing two architectural patterns for dynamically adapting functionality in online software products - best paper award," in *Proceedings of the 5th International Conferences on Pervasive Patterns and Applications (PATTERNS 2013)*, 2013, pp. 20–25.

[2] A. Dubey and D. Wagle, "Delivering software as a service," *The McKinsey Quarterly*, vol. 6, pp. 1–12, 2007.

[3] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, "Multi-tenant databases for software as a service: schema-mapping techniques," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 1195–1206.

[4] S. Jansen, G. Houben, and S. Brinkkemper, "Customization realization in multi-tenant web applications: case studies from the library sector," *Web Engineering*, pp. 445–459, 2010.

[5] K. Pohl, G. Böckle, and F. van der Linden, *Software product line engineering: foundations, principles, and techniques*. Springer-Verlag, New York, 2005.

[6] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, *A pattern language*. Oxford University Press, 1977.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-wesley Reading, MA, 1995, vol. 206.

[8] P. Evitts and D. Hinchcliffe, *A UML pattern language*. Macmillan Technical Publishing, 2000, vol. 201.

[9] D. Mapelsden, J. Hosking, and J. Grundy, "Design pattern modelling and instantiation using dpml," in *Proceedings of the 40th International Conference on Tools Pacific: Objects for internet, mobile and embedded applications*. Australian Computer Society, Inc., 2002, pp. 3–11.

[10] D. Maplesden, J. G. Hosking, and J. C. Grundy, "A visual language for design pattern modelling and instantiation." in *Design Pattern Formalization Techniques*, 2007, pp. 338–339.

[11] A. Lauder and S. Kent, "Precise visual specification of design patterns," in *ECOOP98Object-Oriented Programming*. Springer, 1998, pp. 114–134.

[12] M. Jaring and J. Bosch, "Representing variability in software product lines: A case study," *Software Product Lines*, pp. 219–245, 2002.

[13] J. Bayer, . Gerard, O. Haugen, J. Mansell, B. Møller-Pedersen, J. Oldevik, P. Tessier, J. Thibault, and T. Widen, "Consolidated product line variability modeling," in *Software Product Lines*. Springer, 2006, pp. 195–241.

[14] R. Mietzner, T. Unger, R. Titze, and F. Leymann, "Combining Different Multi-tenancy Patterns in Service-Oriented Applications," in *Proceedings of the IEEE International Enterprise Distributed Object Computing Conference*, 2009, pp. 131–140.

[15] J. Kabbedijk and S. Jansen, "The role of variability patterns in multi-tenant business software," in *Proceedings of the WICSA/ECSA 2012 Companion Volume*. ACM, 2012, pp. 143–146.

[16] M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques," *Software: Practice and Experience*, vol. 35, no. 8, pp. 705–754, 2005.

[17] A. Benlian and T. Hess, "Opportunities and risks of software-as-a-service: Findings from a survey of it executives," *Decision Support Systems*, vol. 52, no. 1, pp. 232–246, 2011.

[18] A. Hevner and S. Chatterjee, *Design research in information systems: theory and practice*. Springer, 2010, vol. 22.

[19] P. Runeson and M. H´ost, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[20] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The*. Pearson Higher Education, 2004.

[21] W. Van der Aalst, A. ter Hofstede, and M. Weske, "Business process management: A survey," *Business Process Management*, pp. 1019–1019, 2003.

[22] B. Carpenter, G. Fox, S. Ko, and S. Lim, "Object serialization for marshalling data in a java interface to mpi," in *Proceedings of the ACM 1999 conference on Java Grande*. ACM, 1999, pp. 66–71.

[23] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley Professional, 2003.

[24] R. Weinreich, T. Ziebermayr, and D. Draheim, "A versioning model for enterprise services," in *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, vol. 2. IEEE, 2007, pp. 570–575.

[25] W. Sun, X. Zhang, C. Guo, P. Sun, and H. Su, "Software as a service: Configuration and customization perspectives," in *Congress on Services Part II*. IEEE, 2008, pp. 18–25.

Advanced Preprocessing of Binary Executable Files and its Usage in Retargetable Decompilation

Jakub Křoustek, Peter Matula, Dušan Kolář, and Milan Zavoral
Faculty of Information Technology, IT4Innovations Centre of Excellence
Brno University of Technology
Brno, Czech Republic
{ikroustek, imatula, kolar}@fit.vutbr.cz, xzavor02@stud.fit.vutbr.cz

***Abstract*—Retargetable machine-code decompilation is used for a platform-independent transformation of executable files into a high level language (HLL) representation (e.g., C language). It is a complex task that must deal with a lot of different platform-specific features and missing information. Accurate preprocessing of input executable files is one of the necessary prerequisites in order to achieve the best results. Furthermore, we can use gathered information to achieve higher quality of decompilation. This paper presents an extended version of our previous system for an accurate code preprocessing. It is implemented as a generic preprocessing system that consists of a precise compiler and packer detector, plugin-based unpacker, converter into an internal platform-independent file format, and debugging information gathering library. We also describe an utilization of the collected information in a problem of automatic data-type reconstruction. This system has been adopted and tested in an existing retargetable decompiler. According to our experimental results, the proposed retargetable solution is fully competitive with existing platform-dependent tools.**

Keywords–reverse engineering, decompilation, packer detection, unpacking, executable file, Lissom.

I. Introduction

This article is closely related to the paper [1]. We extend this previous paper by presenting several new methods of packer and compiler detection (e.g., heuristics-based detection, signatures for ELF file format) and by describing our novel approach of preprocessing in type-recovery phase of decompilation (e.g. exploitation of debugging information, known library function calls). We also present re-evaluation of all experimental tests.

Reverse engineering is used often as an initial phase of a reengineering process. As an example we can mention reengineering of legacy software to operate on new computing platforms. One of the typical reverse-engineering tools is a machine-code decompiler, which reversely translates binary executable files back into an HLL representation, see [2], [3] for more details. This tool can be used for binary code migration, malware analysis, source code reconstruction, etc.

More attention is paid to retargetable decompilation in recent years. The goal is to create a tool capable to decompile applications independent of their origin into a uniform code representation. Therefore, it must handle different target architectures, operating systems, programming languages, and their compilers. Moreover, applications can be also packed or protected by so-called *packers* or *protectors*. This is a typical case of malware. Therefore, such input must be *unpacked* before it is further analyzed; otherwise, its decompilation will be inaccurate or impossible at all. Note: in the following text, we use the term *packing* for all the techniques of executable file creation, such as compilation, compression, protection, etc.

In order to achieve retargetable decompilation, its preprocessing phase is crucial because it eliminates most of the platform-specific differences. For example, this phase is responsible for a precise analysis of an input application (e.g., detection of a target platform). Whenever a presence of a packed code is detected, such application has to be unpacked.

Furthermore, the platform-dependent object file format (OFF) is converted into an internal uniform code representation. The final task of preprocessing is an information gathering, such as detection of originally used programming language, compiler, its version, or detection and processing of debugging information. This information is valuable during the following phases of decompilation because different languages and compilers use different features and generate unique code constructions; therefore, such knowledge implies more accurate decompilation.

In this paper, we present several platform-independent preprocessing methods, such as language and compiler detection, executable file unpacking, conversion, and format-independent debugging information processing. We also demonstrate a utilization of this information on example of the data-type recovery analysis. These methods were successfully interconnected, implemented, and tested in a preprocessing phase of an existing retargetable decompiler developed within the Lissom project [4].

The paper is organized as follows. Section II discusses the related work of executable file preprocessing. Then, we briefly describe the retargetable decompiler developed within the Lissom project in Section III. In Section IV, we give a motivation for a compiler and packer detection within decompilation. Afterwards, our own methods used in the preprocessing phase are presented in Section V. Section VI shows how the data-type recovery analysis uses previously gathered information. Experimental results are given in Section VII. Section VIII closes the paper by discussing future research.

II. Related Work

There are several studies and tools focused on binary executable file analysis and transformation. Most of them are not focused directly on decompilation but some of these ideas

can be applied in this field. Their major limitation for such usage is their bounding to one particular target platform.

In this section, we briefly mention several existing tools used for packer detection, unpacking, OFF conversion and debugging information gathering.

A. Compiler and Packer Detection

The knowledge of the originally used tool (e.g., compiler, linker, packer) for executable creation is useful in several security-oriented areas, such as anti-virus or forensics software [5]. Overwhelming majority of existing tools are limited to the Windows Portable Executable (WinPE) format on the Intel x86 architecture and they use signature-based detection. Almost all of these tools are freeware but not open source.

Formats of signatures used by these tools for pattern matching usually contain a hexadecimal representation of the first few machine-code instructions on the application's entry point (EP). EP is an address of the first executed instruction within the application. A sequence of these first few instructions creates a so-called *start-up* or *runtime* routine, which is quite unique for each compiler or packer and it can be used as its footprint. Accuracy of detection depends on the signature format, their quality, and used scanning algorithm. Identification of sophisticated packers may need more than one signature.

Databases with signatures are either internal (i.e., precompiled in code of a detector), or stored in external files as a plain text. The second ones are more readable and users can easily add new signatures. However, detection based on external signatures is slower because they must be parsed at first. Some detection tools are distributed together with large, third-party external databases.

B. Unpacking

Binary executable file packing is done for one of these reasons—code compression, code protection, or their combination. The idea of code compression is to minimize the size of distributed files. Roughly speaking, it is done by compressing the file's content (i.e., code, data, symbol tables) and its decompression into memory or into a temporal file during execution.

Code protection can be done by a wide range of techniques (e.g., anti-debugging, anti-dumping, insertion of self-modifying code, interpretation of code in internal virtual machine). It is primarily used on MS Windows but support of other platforms is on arise in the last years (e.g., gzexe and Elfcrypt for Linux, VMProtect for Mac OS X, multi-platform UPX and HASP).

Packers are proclaimed to be used for securing commercial code from cracking; however, they are massively abused by malware authors to avoid anti-virus detection. Decompilation of compressed or protected code is practically impossible, mainly because it is "just" a static code analysis and unpacking is done during the runtime. Therefore, it is crucial to solve this issue in order to support decompilation of this kind of code.

UPX is a rare case of packers because it also supports unpacking itself. Unpacking is a very popular discipline of reverse engineering and we can find tools for unpacking many versions of all popular packers (e.g., ASPackDie, tEunlock, UnArmadillo). We can also find unpacking scripts for popular debuggers, like OllyDbg, which do the same job.

Currently, about 80% to 90% of malware is packed [6] and about 10 to 15 new packers are created from existing ones every month [7], more and more often by using polymorhic code generators [8]. In past, there were several attempts to create generic unpackers (e.g., ProcDump, GUW32), but their results were less accurate than packer-specific tools. However, creation of single-purpose unpackers from scratch is a time consuming task. Once again, these unpacking techniques are developed primarily for MS Windows and other platforms are not covered.

C. Object-File-Format Conversion

This part is responsible for converting platform-dependent file formats into an internal representation. We can find several existing projects focused on this task. They are used mostly for OFF migration between two particular platforms and they were hand-coded by their authors just for this purpose. Therefore, they cannot be used for retargetable computing.

A typical example is the MAE project [9], which supports execution of Apple Macintosh applications on UNIX. Sun Microsystems Wabi [10] allows conversion of executables from Windows 3.x to Solaris. AT&T's FreePort Express is another binary translator of SunOS executables into the Digital UNIX format. More examples can be found in [11].

D. Debugging Information Gathering

Debugging information is generated by compilers and traditionally used by debuggers to find and fix software bugs [12]. Since it represents relationship between the machine code and the original source code, it may as well be exploited in decompilation, or any other executable file analysis. The typical use in reverse engineering is to evaluate accuracy of the analysis by comparing inferred results with those from the debugging information.

This approach was used in [13] where the `readelf` utility was used to extract the debugging data, or in [14] and [15] by using the `libdwarf` library. It is however not clear whether any of these tools is capable to incorporate such information to its algorithm and produce more accurate output because of it. Since most of the executable-analysis applications are linked to a particular architecture and platform it is also unlikely that they are able to process different debugging formats.

Despite the fact malware rarely contains such additional data, it would be foolish not to capitalize on them if they are actually present. This also opens new areas of decompiler applications. For example in binary verification of critical programs, which may be intentionally compiled with the debugging information to make analysis easier.

For this reasons, we have already created the debugging information preprocessing libraries for the two most widely used formats (DWARF, Microsoft PDB), and used them for recovery of variables, functions, and arguments [16]. In the original paper [1], we used an untyped Python-like language and we did not exploit the full potential. In this paper we

show how to incorporate precise data types obtained from the debugging information to our type recovery algorithm and propagate them throughout the whole program.

III. Lissom Project's Retargetable Decompiler

The Lissom project's [4] retargetable decompiler aims to be independent on any particular target architecture, operating system, or OFF. It consists of two main parts—the preprocessing part and the decompilation core, see Figure 1. Its detailed description can be found in [17], [18].

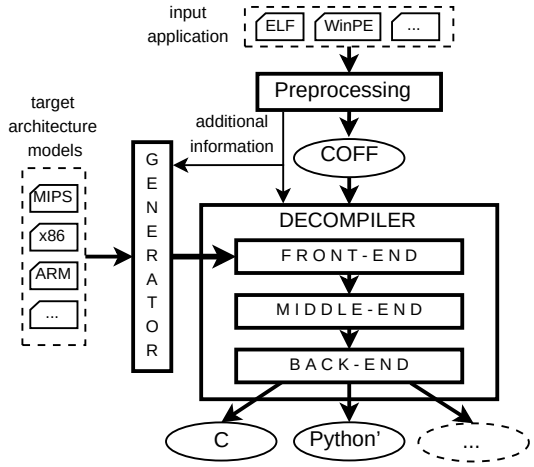


Figure 1. The concept of the Lissom project's retargetable decompiler.

The preprocessing part is described in the following section. Basically, it unpacks and unifies examined platform-dependent applications into an internal Common-Object-File-Format (COFF)-based representation.

Afterwards, such COFF-files are processed in the decompilation core, which is partially automatically generated based on the description of target architecture. This decompilation phase is responsible for decoding of machine-code instructions, their static analysis, recovery of HLL constructions (e.g., loops, functions), and generation of the target HLL code. Currently, the C language and a Python-like language are used for this purpose and the decompiler supports decompilation of MIPS, ARM, and x86 executables.

IV. Motivation

The information about the originally used compiler is valuable during the decompilation process because each compiler generates different code in some cases; therefore, such knowledge may increase a quality of the decompilation results. One of such cases is a usage of so-called instruction idioms. Instruction idiom represents an easy-to-read statement of the HLL code that is transformed by a compiler into one or more machine-code instructions, which behavior is not obvious at the first sight. See [19] for an exhausting list of the existing idioms.

We illustrate this situation on an example depicted as a C language code in Figure 2. This program uses an arithmetical expression "`-(a >= 0)`", which is evaluated as `0` whenever the variable `a` is smaller than zero; otherwise, the result is evaluated as `-1`. Note: the following examples are independent on the used optimization level within the presented compilers. All compilers generate 32-bit Linux ELF executable files for Intel x86 architecture [20] and the assembly code listings were retrieved via `objdump` utility.

```
#include <stdio.h>

int main(int argc, char **argv)
{
   int a;

   scanf("%d", &a);
   // Prints - "0" if the input is smaller than 0
   //        - "-1" otherwise
   printf("%d\n", -(a >= 0));

   return 0;
}
```

Figure 2. Source code in C.

Several compilers substitute code described in Figure 2 by instruction idioms. Moreover, different compilers generate different idioms. Therefore, it is necessary to distinguish between them. For example, code generated by the GNU compiler GCC version 4.0.4 [21] is depicted in Figure 3. As we can see, the used idiom is non-trivial and its readability is far from the original expression.

```
; Address   Hex dump    Intel x86 instruction
;-------------------------------------------
; scanf
; Variable 'a' is stored in %eax
  80483e2:  f7 d0       not  %eax
  80483e4:  c1 e8 1f    shr  $31,%eax
  80483e7:  f7 d8       neg  %eax
; Print result stored in %eax
; printf
```

Figure 3. Assembly code generated by gcc 4.0.4.

The Clang compiler is developed within the LLVM project [22], [23]. Output of this compiler is illustrated in Figure 4. As we can see, Clang uses idiom, which is twice as long as the previous one and it is assembled by the different set of instructions. Therefore, it is not possible to implement one generic decompilation analysis. Such solution will be inaccurate and slow (i.e., detection of all existing idioms no matter on the originally used compiler).

```
; Address   Hex dump        Intel x86 instruction
;-----------------------------------------------
; scanf
; Variable 'a' is stored on stack at -16(%ebp)
  8013bf:   83 7d f0 00     cmpl   $0,-16(%ebp)
  8013c3:   0f 9d c2        setge  %dl
  8013c6:   80 e2 01        and    $1,%dl
  8013c9:   0f b6 f2        movzbl %dl,%esi
  8013cc:   bf 00 00 00 00  mov    $0,%edi
  8013d1:   29 f7           sub    %esi,%edi
  8013d3:   ;...
  8013d6:   89 7c 24 04     mov    %edi,4(%esp)
; Print result stored on stack at 4(%esp)
; printf
```

Figure 4. Assembly code generated by clang 3.1.

Decompilation of instruction idioms (or other similar constructions) produces a correct code; however, without any compiler-specific analysis, this code is hard to read by a human because it is more similar to a machine-code representation than to the original HLL code. Compiler-specific analyses are focused on these issues (e.g., they detect and transform idioms back to a well-readable representation), but the knowledge of the originally used compiler and its version is mandatory.

Figure 5 depicts decompilation results for the gcc compiled code listed in Figure 3 (i.e., code generated by gcc 4.0.4). The Lissom retargetable decompiler was used for this task. As we can see, the expression contains bitwise shift and xor operators instead of the originally used comparison operator. This makes the decompiled code hard to read.

```
#include <stdint.h>
#include <stdio.h>

int main(int argc, char **argv)
{
   int apple;
   apple = 0;
   scanf("%d", &apple);
   printf("%d\n", -(apple >> 31 ^ 1));
   return 0;
}
```

Figure 5. Decompiled source code from program listed in Figure 3. In this case, the decompiler lacks any compiler-specific analysis and the result is hard to read.

Furthermore, it is important to detect compiler version too. In Figure 6, we illustrate that the different versions of the same compiler generate different code for the same expression. We use gcc version 3.4.6 and the C code from the Figure 2.

```
; Address  Hex dump            Intel x86 instruction
;-------------------------------------------------
; scanf
; Variable 'a' is stored on stack at -4(%ebp)
  80483f3: 83 7d fc 00         cmpl $0,-4(%ebp)
  80483f7: 78 09               js   8048402
  80483f9: c7 45 f8 ff ff...   movl $-1,-8(%ebp)
  8048400: eb 07               jmp  8048409
  8048402: c7 45 f8 00 00...   movl $0,-8(%ebp)
  8048409:
; Print result stored on stack at -8(%ebp)
; printf
```

Figure 6. Assembly code generated by gcc 3.4.6.

In this assembly code snippet, we can see that no instruction idiom was used. The code simply compares the value of a variable with zero and sets the result in a human-readable form. It is clear that the difference between the code generated by the older (Figure 6) and the newer version (Figure 3) of this compiler is significant. Therefore, we can close this section stating that information about the used compiler and its version is important for decompilation.

V. Preprocessing Phase of the Retargetable Decompiler

In this section, we present a design of the preprocessing phase within the Lissom project retargetable decompiler. The complete overview is depicted in Figure 7. The concept consists of the following parts.

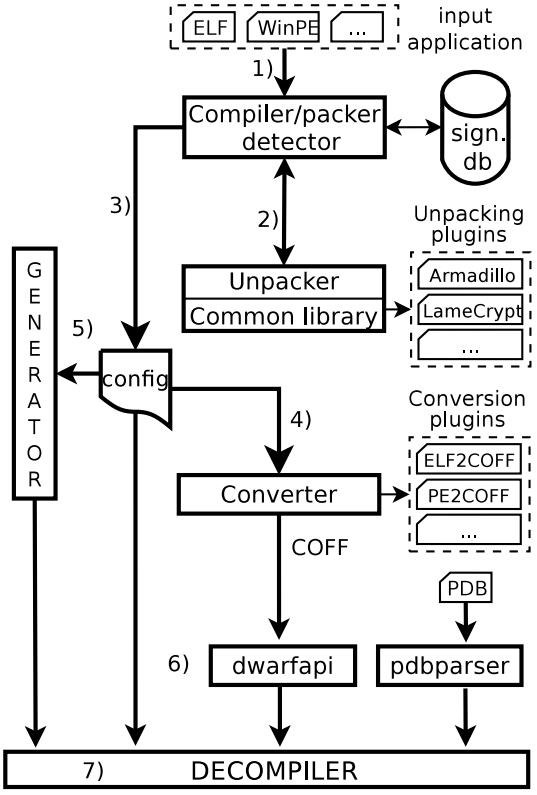


Figure 7. The concept of the preprocessing phase.

At first, the input executable file is analyzed and the used OFF is detected. All common formats are supported (e.g., WinPE, UNIX ELF, Mach-O). Information about the target processor architecture is extracted from the OFF header (e.g., `e_machine` entry in ELF OFF) and it is used together with other essential information in further steps.

The next part of this step is a detection of a tool used for executable creation. This is done by using a signature-based detection of start-up code as described in Section II. Example of such a start-up code can be seen in Figure 8. Signature for this code snippet is "`5589E583EC18C7042401000000FF15--------E8`", where each character represents a nibble of instruction's encoding. All variable parts must be skipped during matching by a wild-card character "`-`", e.g., a target address in the `call` instruction. This signature format is quite similar to formats used by other detectors listed in Section VII.

```
; Address    Hex dump        Intel x86 instruction
; ------------------------------------------------
  0040126c:  55              push %ebp
  0040126d:  89e5            mov  %esp, %ebp
  0040126f:  83ec18          sub  $0x18, %esp
  00401272:  c7042401000000  movl $0x1, (%esp)
  00401279:  ff1500000000    call *0x0
  0040127f:  e8              ...
```

Figure 8. Start-up code for MinGW gcc v4.6 on x86 (`crt2.o`) generated by `objdump -d`.

Our signature format also supports two new features—description of nibble sequences with zero or more occurrences and description of unconditional short jumps. Example of the former one is "(90)", denoting an optional sequence of nop instructions for x86 architecture. Example for the second one is "#EB", denoting an unconditional short jump for the same architecture, which size is specified in the next byte; everything between the jump and its destination is skipped. In Figure 9, we can find a code snippet covered by signature "#EB(90)40".

```
; Address    Hex dump         Intel x86 instruction
;-----------------------------------------------
  00401000:  eb 02            jmp short <00401004>
  00401002:  xx xx            ; don't care
  00401004:  90               nop
  00401005:  90               nop
  00401006:  40               inc %eax
```

Figure 9. Example of advanced signature format.

These features come handy especially for polymorphic packers [8] producing a large number of different start-up codes (e.g., Obsidium packer). Describing one version of such packer usually needs dozens of classical signatures. However, this number can be significantly reduced by using the above-mentioned features.

Signatures within our internal database were created with focus on the detection of the packer's version. This information is valuable for decompilation because two different versions of the same packer may produce diverse code constructions. The database also contains signatures for non-WinPE platforms; therefore, it is not limited like most of other tools. Finally, new signatures can be automatically created whenever the user can provide at least two files generated by the same version of packer. Presence of multiple files is mandatory in order to find all variable nibbles in the start-up code.

Unfortunately, some polymorphic packers (e.g., Morphine encryptor) cannot be described via extended format of signatures. These packers generate entirely different start-up routine for each packed file. For instance, there is an example of three different start-up routines generated by the Morphine encryptor:

```
1C1C26083EB00F6DFF6DF535BFC7C03C1EC005
7408525566C1C4105D5A51510AC95959F9FC60
510FB6C9770525FFFFFFFFF8E2F35983FA2D8B
```

If we use these samples to create signature, we get the following result:

```
--------------------------------------
```

As we can see, resulting signature contains only wild-card characters, which is useless for detection. Therefore, in order to support a precise detection, we support concept of additional heuristics that are focused on polymorphic packers. These heuristics are analysing several properties of the executable file (e.g., attributes of sections, information stored within file header, offset of EP in executable file) and they perform the detection based on a packer-specific behavior. Example of such heuristic is illustrated in Figure 10—it takes into account file format, target architecture, offset of EP in file, and information about file sections. Heuristic is written in C++-like pseudocode.

```
if(file_format == WIN_PE &&
   target_architecture == INTEL_X86 &&
   EP_file_offset >= 0x400 &&
   EP_file_offset <= 0x1400 &&
   sections[0].name == ".text" &&
   sections[1].name == ".data" &&
   sections[2].name == ".idata" &&
   sections[2].size == 0x200)
{
   return "Morphine 1.2";
}
```

Figure 10. Heuristic for Morphine encryptor v1.2.

Except of heuristics for precise detection of polymorphic packers, we also support simpler heuristics, which are focused only on a name, number, and order of sections. Such heuristics cannot detect exact version of used packer, but they are useful if signature database does not contain entry for related tool. Their overview is depicted in Table I.

Whenever a usage of packer is detected in the first phase, the unpacking part is invoked. Unpacking is done by our own generic unpacker, which consists of a common unpacking library and several plugins implementing unpacking of particular packers. The common library contains the necessary functions for rapid unpacker creation, such as detection of the original entry point (OEP), dump of memory, fixing import tables, etc. Therefore, a plugin itself is very tiny and contains only code specific to a particular packer.

A plugin can be created in two different ways: either it can reverse all the techniques used by the packer and produce the original file, or the plugin can execute the packed file, wait for its decompression, and dump its unprotected version from memory to file. The first one is hard to create because it takes a lot of time to analyze all the used protection techniques. Its advantage is that unpacking can be done on any platform because the file is not being executed. That is the main disadvantage of the second approach. Such a plugin can be created quickly; however, it must be executed on the same target platform. In present, we support unpacking of several popular packers like Armadillo, UPX (Linux and Windows), NoodleCrypt and others in the second way. See Section VIII for its future research.

After unpacking, the re-generated executable file is once more analyzed. In rare cases, second packer was used and we need to unpack this file once more. Otherwise, the analysis will try to detect the used compiler and its version, and generate a configuration file, which is used by other decompilation tools. This configuration file also contains information about the target architecture, endianness, bitwidth, address of OEP, etc.

Afterwards, the platform-specific unpacked executable file is converted into an internal COFF-based representation. The converter is also implemented in a plugin-based way and each plugin converts one particular OFF. Currently, we support ELF, WinPE, Mach-O, and several others OFF. See [11] for more details about this tool.

TABLE I. OVERVIEW OF HEURISTICS FOCUSED ON THE NAME AND ORDER OF SECTIONS.

packer	heuristic
Upack	`sections[0].name == ".Upack"`
PE-PACK	`sections[last].name == ".PEPACK!!"`
WWPack32	`sections[last].name == ".WWP32"`
yoda's Crypter	`sections[last].name == "yC"`
LameCrypt	`sections[last].name == "lamecryp"`
ASPack	`sections[last - 1].name == ".aspack" && sections[last].name == ".adata"`
PEBundle	`sections[last - 1].name == "pebundle" && sections[last].name == "pebundle"`
UPX	`numberOfSections == 3 && sections[0].name == "UPX0" && sections[1].name == "UPX1" && (sections[2].name == "UPX2" || sections[2].name == ".rsrc")`
Petite	`numberOfSectionsWithName(".petite") == 1`
PKLite	`numberOfSectionsWithName(".pklstb") == 1`
Krypton	`numberOfSectionsWithName(".krypton") == 1 && numberOfSectionsWithName("YADO") >= 1`
NFO	`numberOfSectionsWithName("NFO") == numberOfSections`
PELock NT	`numberOfSectionsWithName("PELOCKnt") > 0 && (numberOfSectionsWithName("PELOCKnt") >= numberOfSections - 2 || numberOfSections == 1)`
PELock v1.x	`numberOfSectionsWithName(".pelock") > 0 && numberOfSectionsWithName(".pelock") >= numberOfSections - 1`
MEW v10	`numberOfSections == 2 && section[0].name == ".data" && section[1].name == ".decode"`
MEW v11 SE 1.x	`numberOfSections == 2 && section[0].name.containString("MEW")`
NsPack v2.x	`for(i = 0; i < numberOfSections; ++i) sections[i].name == string("nsp" + numToStr(i))`
NsPack v3.x	`for(i = 0; i < numberOfSections; ++i) sections[i].name == string(".nsp" + numToStr(i))`

Using the information about the target architecture in the configuration file, the instruction decoder is automatically created by the generator tool [18]. Instruction decoder is the first part of the decompiler's front-end, which translates machine code instructions into a semantics description of their behavior.

The last step before the actual decompilation is processing of debugging information (if present). Currently we support two major debugging formats, architecture independent DWARF and Microsoft PDB. Each of them is handled by a separate specialized library that loads their contents to the internal representation and provides convenient access methods. Thanks to the previous steps, it is possible to determine input's platform and check for the presence of the DWARF sections in the COFF file.

Since the PDB debugging information is distributed in a separate file, checking object file would be pointless and it is necessary to provide such file whenever an additional information has to be used. DWARF format is preprocessed by the mid-layer library called `dwarfapi`. It uses another library named `libdwarf` to parse low-level debugging information, upon which it builds high-level, object-oriented data structures.

Because there is no available PDB toolkit that would suit our needs, we created our own parser library called `pdbparser`. Basic principles behind both of these tools were described in [16]. Since then, we further extended them to support all data types present in the C programming language. Adding object-oriented features used in the C++ and other similar languages is planned in the near future.

Finally, the COFF executable file is processed in the generated decompiler according to the configuration file. Using the provided information about used compiler, the decompiler can selectively enable compiler-specific analyses (e.g., detection of instruction idioms, recovery of functions). One of the first steps is to check for the debugging information presence and load it to the internal canonical representation using already described libraries. This way, any further analysis can access this information in an unified manner no matter the format of an underlying source.

VI. PREPROCESSING IN THE TYPE RECOVERY ANALYSIS

The goal of a type recovery analysis is to associate each piece of data with a high-level data type as close to the original source code type as possible. We presented the design of a data-flow based type recovery algorithm used by our retargetable decompiler in [24]. Simplified overview is depicted in Figure 11.

Reconstruction can be divided into the three main phases: (1) Object (registers, global/local variables, function arguments, etc.) initialization, where each occurrence gets an initial type, and these types are interconnected by the propagation equations. (2) Simple and composite analysis run over the set of objects and equations. Objects' types are inferred based on their initial types and the semantics of the operations they occurred in. Analysis ends once the system's fixpoint is reached. (3) Reconstructed types are used in the output intermediate representation (IR). The original paper focused solely on the core of the analysis – simple data-type inference based on the semantics of the individual instructions.

This approach can be applied to any input without additional conditions. The disadvantage is its lower accuracy compared to the other potential type information sources. This section presents utilization of two highly accurate data-type sources use of which is enabled by the extensive preprocessing.

A. Data-Type Debugging Information

Debugging information contains exact types of all objects existing in the original source code. By the time of the type recovery analysis run, they have already been preprocessed and are easily accessible. All that needs to be done is to apply

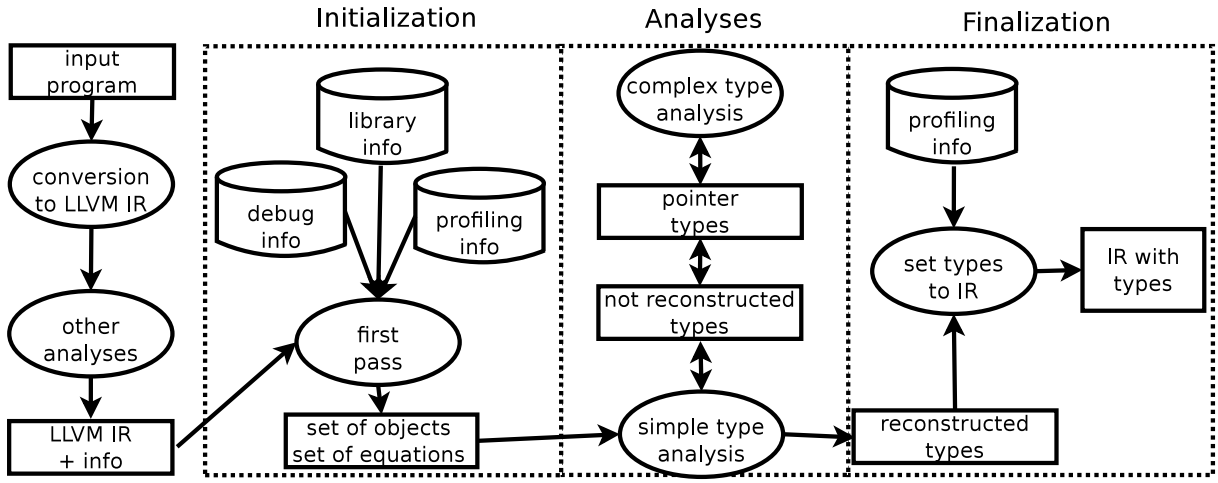


Figure 11. Data-type recovery analysis scheme. Taken from [24].

them in an initialization phase instead of inferring initial types from the nature of object's occurrence. This may look simple enough, but we have to make sure that the type will be assigned to the correct object. If the debugging information is present, functions and their arguments are reconstructed precisely and it is trivial to link them with their true types.

```
#include <stdlib.h>
#include <stdio.h>

struct s { int i; char c; float f; };
struct s global;

int main()
{
   FILE *pFile;
   pFile = fopen("some.file", "w+");

   float local = rand();
   fprintf(pFile, "%f", local);

   fscanf(pFile, "%d %c %f",
          global.i, global.c, global.f);
   printf("%d %c %f",
          global.i, global.c, global.f);

   return 0;
}
```

Figure 12. Example of the original source code used for demonstration of the type recovery.

For global variables, things gets slightly complicated. Variable analysis detected all the global memory accesses, whose addresses can be statically determined. Then it created simple global variables for these addresses and used their names in the access instructions. For example, variable `global` in Figure 12 was recognized from three different accesses to its elements as three separate variables. The first one at address X, the second at $X + 4$ and the third at $X + 8$.

As is displayed in Figure 13 (showing relevant fragment of the DWARF data), debugging information contains entry only for the entire composite object. Fortunately, it is not complicated to link the computed address X with the address from the debug data (`0x0891daa4` in this case) and assign the true type to the first simple variable.

```
DW_TAG_subprogram
  DW_AT_name              "main"
  DW_AT_frame_base <loclist with 3 entries follows>
    [0]<lowpc=0x0000><highpc=0x0004> DW_OP_reg29
    [1]<lowpc=0x0004><highpc=0x0010> DW_OP_breg29+24
    [2]<lowpc=0x0010><highpc=0x0104> DW_OP_breg30+24

DW_TAG_variable
  DW_AT_name              "local"
  DW_AT_location          DW_OP_fbreg -24

DW_TAG_variable
  DW_AT_name              "global"
  DW_AT_location          DW_OP_addr 0x0891daa4
```

Figure 13. Relevant fragment of the DWARF debugging information generated for the code in Figure 12.

Based on the known type size, the algorithm can merge all subsequent simple variables into one composite object. To achieve the code quality similar to the original source, all instructions accessing simple globals must be changed to operations reading or writing the corresponding composite elements. Using this method, we are able to assign the true types even to the objects accessed by addresses, whose values cannot be statically computed (e.g., accessing global array in a cycle using an iterator). However, creating correct access instructions with iterators demands usage of the composite type recovery analysis, which is beyond the scope of this article.

The same principles used for the globals can be applied to the stack (local) objects as well. However, linking types to the related objects gets much more complicated in this case.

Looking at the same examples as before, we can see that stack object `local` is located at `DW_OP_fbreg -24`. This means that it is at offset of `-24` from the current frame base. The frame base is determined by the actual program counter and the corresponding expression in the function's location list. For example, if program counter is between `0x0010` and `0x0104`, then frame base is equal to `DW_OP_breg30 + 24`, where `DW_OP_breg30` represents current value of the register labeled by DWARF with the number `30`.

To successfully use this information, we need to make sure our stack analysis computes the same stack variable offsets as would be calculated by the debugger using the debugging data. Furthermore, we need to be able to repeat computation of the `DW_AT_location`. To do so, mapping between the DWARF register numbers and real architecture registers must be known. If stack offsets were computed correctly, it is possible to link them with the `DW_AT_location` results and find corresponding DWARF entries for the detected stack variables. Finally, it is trivial to apply true types and perform the same kind of aggregation and instruction replacement as for the global variables.

Note: Exploitation of the PDB debugging information is similar or sometimes even easier to the presented DWARF usage.

B. Known Library Function Calls

Calls of the known library functions are another highly accurate data-type source whose optimal usage is enabled by the extensive input preprocessing. It is providing types of the same quality as the debugging data without the need of any additional information in the executable. However, decompiler must be able to detect linked function calls and have the database of functions' prototypes containing information about the types they use. Generic signature based function code detection and library type information (LTI) file creation is described in [25]. In this paper, we deal with its application in the type recovery algorithm and the advantages gained from the preprocessing.

```
%struct.struct_drand48_data =
  type { [3 x i16], [3 x i16], i16, i16, i64 }

# FILE * fopen (const char *name, const char *mode)
  fopen %struct.struct__IO_FILE* 2 i8*, i8*

# int fscanf (FILE *stream, const char *form, ...)
  fscanf i32 3 %struct.struct__IO_FILE*, i8*, ...

# void * malloc (size_t size)
  malloc void* 1 i32

# double strtod (const char *str, char **ptr)
  strtod double 2 i8*, OUT REF i8**
```

Figure 14. Examples of the library type information entries from *stdlib.lti* and *stdio.lti* files.

Separate LTI file containing the function prototypes and the definitions of used composite types is generated for each known standard library. Figure 14 depicts few real examples from the *stdio.h* and *stdlib.h*. Lines starting with the symbol `#` are comments, other lines are actual entries written in the LLVM IR syntax. The first record is an example of a structure definition containing two arrays and three simple members. Other lines show prototypes of some well-known functions.

Function name is always the first, followed by the function's return type, number of arguments and finally arguments' types. If the function is variadic, its parameter list ends with the `...` token. Arguments may be also flagged to express some additional information about their typical use. For example, `OUT` signals that something is returned through the parameter, `REF` means that argument is typically passed by the reference. This makes it possible to decide, which of the different call variants of `strtod()` depicted in Figure 15 is more likely to be used.

Thanks to the input preprocessing, it is possible to pick the optimal set of LTI files matching program's architecture, operating system, and compiler. These LTI files are used to assign the true data-types to argument and return objects each time known function call is detected. Subsequent type propagation will spread the information between other object occurrences and to all objects in its equivalence class (defined by the relation: *to have the same data type*).

```
char in[] = "365.24 29.53";

// Variant #1:
  char* pEnd;
  double d = strtod (in, &pEnd);

// Variant #2:
  char** pEnd;
  double d = strtod (in, pEnd);
```

Figure 15. Several possible variants how to call `strtod()`.

C. Analysis Modification

Beside the original source code objects, decompiler's output usually contains other variables arising from the use of registers or auxiliary local/global variables. Types of such objects are not present in a potential debugging information; therefore, they cannot be set directly. Data types recovered from the function calls are initially set only to the immediate objects used by the call. For this reasons, no matter the type sources quality, decompiler always performs full data-flow type propagation. The only difference is, that some initial object types set in the analysis initialization are more precise than others.

The analysis core presented in [24] infers types by repeated application of the propagation rules and the join function. For each object, the greatest lower bound of all its occurrences is found. Algorithm described in the article takes all initial type estimates as equal and may refine them in order to unify all object's occurrences. Since types obtained from the sources shown in this paper are already precise, this behavior is undesired for their propagation.

The solution is to tag each type with the identification of its origin. More precise the origin, greater the priority during propagation between object's occurrences and other objects in the equivalence class. Types with high enough tags are also saved from any modifications, so that already correct types are not broken in the process.

Origin tags in an ascending order of precision are: (1) Default type, 32-bit signed integer. (2) Type inferred from the instruction semantics. (3) Non-default type that was set by some previous analysis. (4) Type from the dynamic analysis. (5) Type from the known function call. (6) Type from the debugging information. Only the last two are saved from any modifications. It is however possible, in some special cases, to further enhance the final outcome. Allocation related functions are the typical example. As we can see in Figure 14, return type

of the `malloc()` function is pointer to `void`. However, this cannot be the type of the variable where the result is stored. In this case, analysis puts together types from two sources of different precision to get one final data type.

VII. Experimental Results

This section contains an evaluation of the previously described methods of packer detection. The accuracy of our tool (labeled as "Lissom") is compared with the latest versions of existing detectors. Their short overview is depicted in Table II. Detectors are compared in three test sets, see below: (A) WinPE packers, (B) WinPE polymorphic packers, (C) ELF packers and compilers. This section also contains discussion about accuracy of type recovery analysis (D).

TABLE II. Overview of existing compiler/packer detection tools.

tool		signatures		
name	version	internal	external	total
Lissom	1.5	2282	0	2282
RDG Packer Detector [26]	0.7.2	?	10	?
ProtectionID (PID) [27]	0.6.5.5	543	0	543
Exeinfo PE [28]	0.0.3.4	718	7076	7794
Detect It Easy (DiE) [29]	0.81	?	2100	?
NtCore PE Detective [30]	1.2.1.1	0	2806	2806
FastScanner [31]	3.0	1605	1832	3437
PEiD	0.95	672	1774	2446

All of these detection tools use the same approach as our solution—detection using signature matching. As we can see in Table II, most of them use a combination of precompiled internal signatures and a large external database created by the user community. The competitive solutions (except of tool DiE) are limited to WinPE OFF and a number of their signatures varies between hundreds and thousands. The number of internal signatures is not always absolutely precise because some authors do not specify this number, like RDG or FastScanner. Therefore, we had to analyze such applications and try to find their databases manually (e.g., using reverse engineering). We were unable to find it in the RDG and DiE detectors. Our solution consists of 2282 internal signatures for all supported OFFs and we also support the concept of external signatures.

By using reverse engineering, we also figured out that several tools (e.g., PEiD, RDG) use additional heuristic technique for packer detection. These techniques are similar to our solution descripted in Section V. Using this heuristic analysis, PEiD and RDG detectors are able to detect polymorphic packers like Morphine encryptor. However, our solution achieved more accurate results in tests focused on polymorphic packers.

A. WinPE Packers

In total, 40 WinPE packers (e.g., ASPack, FSG, UPX, PECompact, EXEStealth, MEW) and several their versions (107 different tools in total) were used for comparison of previously mentioned detectors. We used these packers for packing several compiler-generated executables—with different size (50kB to 5MB), used compiler, compilation options, and packer options. The purpose is that some packers create different start-up code based on the file size and characteristics (data-section size, PE header flags, etc.). The test set consists of 5317 executable files in total. We prepared three test cases for the evaluation of the proposed solution.

At first, we evaluated the detection of packer's name. This type of detection is the most common and also the easiest to implement because generic signatures can be applied (i.e., signatures with only few fixed nibbles describing complete packer family). On the other hand, this information is critical for the complete decompilation process because if we are unable to detect usage of executable-file protector, the decompilation results will be highly inaccurate. The results of detection are compared in Figure 16.

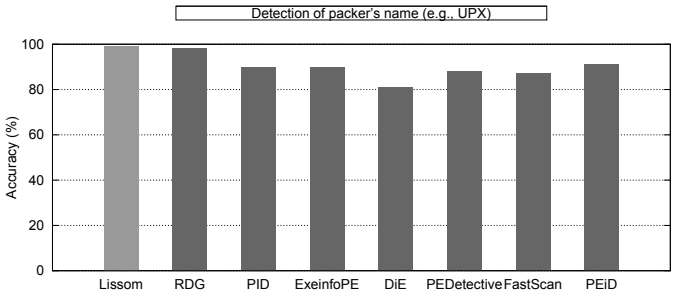


Figure 16. Summary of packer detection (packer names).

According to the results, our tool has the best ratio of packer's name detection (over 99%), while the RDG [26] detector was second with ratio 98%. All other solutions achieved comparable results—between 80% and 91%. We can also notice that larger signature databases do not imply better results in this cathegory (e.g., Exeinfo PE). Such large databases are hard to maintain and they can produce several false-positive results because of too much generic signatures.

Afterwards, we tested the accuracy of tool's major version detection. In other words, this test case was focused on tool's ability to distinguish between two generations of the same tool (e.g., UPX v2 and UPX v3). This feature comes handy in the front-end phase during compiler-specific analysis. For example, the compiler may use in its newer versions more aggressive optimizations that have a very specific meaning and they need a special attention by the decompiler (e.g., instruction idioms, loops transformation, jump tables), see Section IV for details. The results are depicted in Figure 17.

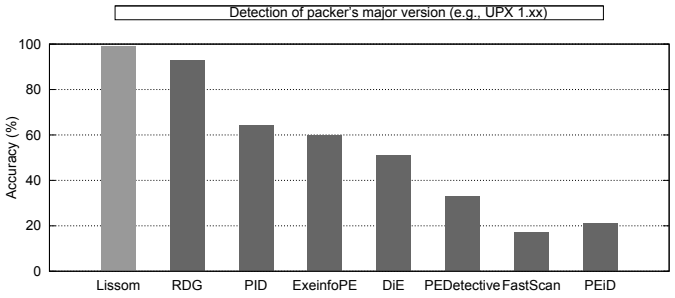


Figure 17. Summary of packer detection (packer versions).

Within this test case, our solution and RDG once again achieved the best results (Lissom scored 99%, RDG scored 93%). None of the programs has exceeded the limit of 80%. Only ExeinfoPE and ProtectionID exceeded 60% success ratio from the others.

Finally, we tested the ratio of precise packer's version detection. This task is the most challenging because it is

necessary to create one signature for each particular version of each particular packer. This information is crucial for the unpacker because the unpacking algorithms are usually created for one particular packer version and their incorrect usage may lead to a decompilation failure.

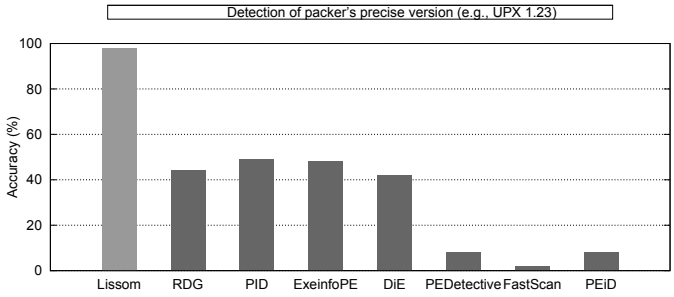


Figure 18. Summary of packer detection (detailed detection).

Based on the results depicted in Figure 18, our detector achieved the best results in this category with 98% accuracy. The results of other solutions were much lower (50% at most). This is mainly because we focus primarily on detecting the precise version and we also support search in the entire PE file and its overlay and not just on its entry point.

B. WinPE Polymorphic Packers

Second test suite is focused on WinPE polymorphic packer labelled as Morphine encryptor. We used two versions of this tool (v1.2 and v2.7). The test set consist of 339 executable files in total and we used the same testing methodology as in the previous case. Only three detectors (Lissom, RDG, and PEiD) are able to detect this packer. Therefore, other detectors were excluded from the results. The results are depicted in Table III.

TABLE III. RESULTS FROM TESTING OF DETECTION OF MORPHINE ENCRYPTOR.

tool	type of detection test		
name	name	major version	detailed version
Lissom	100%	100%	100%
RDG	94.69%	27.73%	27.73%
PEiD	59.88%	59.88%	35.10%

As we can see, our heuristics detection is the most successful (with ratio 100% in all cases). RDG detector exceeded 90% success ratio in detection of packer name, but in other cases its results are poor. Not even PEiD has achieved good results.

C. ELF Packers and Compilers

Last test suite for compiler/packer detectors is focused on detection of ELF compilers (e.g., GCC) and packers (ELFCrypt, UPX)—we used 18 tools and 197 files in total. Only two detectors (Lissom and DiE) supports processing of ELF OFF files. Thus, only these detectors were tested. The results are compared in Table IV.

Even in this test suite, our tool had the most accurate results in all cases. Poor performance of DiE detector in two from three test categories is probably caused by a small number of signatures for ELF OFF.

TABLE IV. RESULTS FROM TESTING OF DETECTION OF ELF COMPILERS AND PACKERS.

tool	type of detection test		
name	name	major version	detailed version
Lissom	98.98%	98.98%	87.31%
DiE	69.04%	4.57%	4.57%

D. Accuracy of type recovery analysis

Figure 19 shows the comparison of the data-type detection accuracy with and without usage of library function call prototypes. Graph does not contain column with the debugging information precision since it is used as reference for other two type sources and it would always be 100% accurate. The set of 26 real world programs written in the C programming language was compiled by the *gcc* compiler for four architectures (MIPS, x86, ARM, Thumb) and four optimization levels (O0 through O3). Debugging information generation was also enabled.

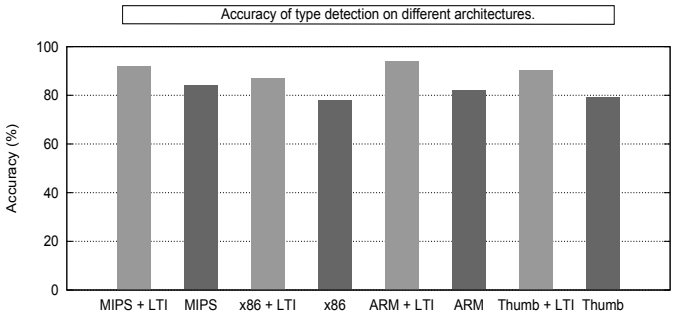


Figure 19. Summary of data-type detection.

All the resulting binaries were subsequently decompiled three times. Data-type recovery was allowed to use different type sources each time (dynamic analysis was not considered or used): (1) Any type source including debugging information. Results served as reference. (2) Enabled library function usage, but no debugging information. First column for each architecture. (3) Only type inference from the instruction semantics or as a result of some previous analysis. Second column for each architecture.

Conservative metric as described in [24] was used to determine data-type accuracy. Average precision rate for each architecture was computed from all combinations of input files and optimizations. We can see that the exploitation of library type information significantly improves type accuracy across all platforms. The improvement is all the more important given that it often provides exact reconstruction of complex well-known structures, which would never be possible just by the static type inference.

VIII. CONCLUSION

This paper was aimed on architecture-independent preprocessing methods used within the existing retargetable decompiler. We introduced methods of packer detection, unpacking, OFF conversion, and debugging information processing. Moreover, we have shown their benefits on precise data-type recovery. Up to now, this concept has been successfully tested on the MIPS, ARM, and x86 architectures within the Lissom project's [4] retargetable decompiler.

We made several tests focused on accuracy of our solution and according to the experimental results, it can be seen that our concept is fully competitive with other existing tools. Our solution achieved more than 98% accuracy in all test cases focused on packer and compiler detection, which was the best result of all examined tools.

We close the paper by proposing three areas for future research. (1) The unpacking phase can be enhanced by using retargetable simulators [32]. Such tools can emulate the target host system and, therefore, it will not be necessary to unpack executables on the same system as its origin. (2) We can further increase decompilation results by creation of new signatures, heuristics, and compiler-specific analyses (e.g., better loop statement recovery, detecting different types of function calls). The process of heuristics creation can be also based on a machine-learning approach. (3) The decompilation results can be increased by extending support on C++ object-oriented debugging information and creation of new function type libraries.

ACKNOWLEDGMENTS

This work was supported by the BUT FIT grant FIT-S-14-2299, and by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

REFERENCES

[1] J. Křoustek and D. Kolář, "Preprocessing of binary executable files towards retargetable decompilation," in 8th International Multi-Conference on Computing in the Global Information Technology (ICCGI'13). Nice, FR: International Academy, Research, and Industry Association (IARIA), 2013, pp. 259–264.

[2] C. Cifuentes, "Reverse compilation techniques," Ph.D. dissertation, School of Computing Science, Queensland University of Technology, Brisbane, QLD, AU, 1994.

[3] M. J. V. Emmerik, "Static single assignment for decompilation," Ph.D. dissertation, University of Queensland, Brisbane, QLD, AU, 2007.

[4] Lissom, http://www.fit.vutbr.cz/research/groups/lissom/, 2013.

[5] G. Taha, "Counterattacking the packers," in Anti-Virus Asia Researchers Conference (AVAR'07), 2007.

[6] T. Brosch and M. Morgenstern, "Runtime packers: The hidden problem?" in Black Hat, 2006.

[7] K. Babar and F. Khalid, "Generic unpacking techniques," in 2nd International Conference on Computer, Control and Communication, 2009, pp. 1–6.

[8] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo, "On the infeasibility of modeling polymorphic shellcode," in 14th ACM Conference on Computer and Communications Security (CCS'07). ACM, 2007, pp. 541–551.

[9] Apple Inc., "Macintosh application environment," http://www.mae.apple.com, 1994.

[10] P. Hohensee, M. Myszewski, and D. Reese, "Wabi cpu emulation," Hot Chips VIII, 1996.

[11] J. Křoustek, P. Matula, and L. Ďurfina, "Generic plugin-based convertor of executable file formats and its usage in retargetable decompilation," in 6th International Scientific and Technical Conference (CSIT'11). Ministry of Education, Science, Youth and Sports of Ukraine, Lviv Polytechnic National University, Institute of Computer Science and Information Technologies, 2011, pp. 127–130.

[12] J. Rosenberg, How Debuggers Work: Algorithms, Data Structures, and Architecture. New York, US-NY: John Wiley, 1996.

[13] E. N. Troshina and A. V. Chernov, "Using information obtained in the course of program execution for improving the quality of data type reconstruction in decompilation," Programming and Computer Software, 2010, pp. 343–362.

[14] J. Lee, T. Avgerinos, and D. Brumley, "Tie: Principled reverse engineering of types in binary programs." in NDSS. The Internet Society, 2011. [Online]. Available: http://dblp.uni-trier.de/db/conf/ndss/ndss2011.html#LeeAB11

[15] K. ElWazeer, K. Anand, A. Kotha, M. Smithson, and R. Barua, "Scalable variable and data type detection in a binary rewriter," SIGPLAN Not., vol. 48, no. 6, Jun. 2013, pp. 51–60. [Online]. Available: http://doi.acm.org/10.1145/2499370.2462165

[16] J. Křoustek, P. Matula, J. Končický, and D. Kolář, "Accurate retargetable decompilation using additional debugging information," in 6th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE'12). International Academy, Research, and Industry Association (IARIA), 2012, pp. 79–84.

[17] L. Ďurfina, J. Křoustek, P. Zemek, D. Kolář, T. Hruška, K. Masařík, and A. Meduna, "Design of a retargetable decompiler for a static platform-independent malware analysis," International Journal of Security and Its Applications (IJSIA), vol. 5, no. 4, 2011, pp. 91–106.

[18] L. Ďurfina, J. Křoustek, P. Zemek, and B. Kábele, "Detection and recovery of functions and their arguments in a retargetable decompiler," in 19th Working Conference on Reverse Engineering (WCRE'12). Kingston, ON, CA: IEEE Computer Society, 2012, pp. 51–60.

[19] H. Warren, Hacker's Delight. Boston, US-MA: Addison-Wesley, 2003.

[20] Intel Corporation, "Intel 64 and IA-32 architectures software developer's manual volume 1: Basic architecture," 2013, http://download.intel.com/products/processor/manual/253665.pdf.

[21] GCC: the GNU Compiler Collection, http://gcc.gnu.org/, 2014.

[22] Clang: A C Language Family Frontend for LLVM, http://clang.llvm.org/, 2013.

[23] The LLVM Compiler Infrastructure, http://llvm.org/, 2013.

[24] P. Matula and D. Kolář, "Reconstruction of simple data types in decompilation," in 4th International Masaryk Conference for Ph.D. Students and Young Researchers (MMK 2013), 2013, pp. 1–10. [Online]. Available: http://www.fit.vutbr.cz/research/view_pub.php?id=10486

[25] L. Ďurfina and D. Kolář, "Generic detection of the statically linked code," in Proceedings of the Twelfth International Conference on Informatics INFORMATICS 2013. Faculty of Electrical Engineering and Informatics, University of Technology Košice, 2013, pp. 157–161. [Online]. Available: http://www.fit.vutbr.cz/research/view_pub.php?id=10461

[26] RDG Packer Detector, http://www.rdgsoft.net/, 2014.

[27] ProtectionID, http://pid.gamecopyworld.com/, 2014.

[28] ExeinfoPE, http://exeinfo.atwebpages.com/, 2014.

[29] DiE, http://www.ntinfo.biz/index.php/detect-it-easy/, 2014.

[30] NtCore PE Detective, http://www.ntcore.com/, 2014.

[31] FastScanner, http://www.at4re.com/, 2014.

[32] Z. Přikryl, "Advanced methods of microprocessor simulation," Ph.D. dissertation, Brno University of Technology, Faculty of Information Technology, 2011.

Towards High Quality Mobile Applications: Android Passive MVC Architecture

Karina Sokolova*†, Marc Lemercier*

*University of Technology of Troyes
Troyes, France
{karina.sokolova, marc.lemercier}@utt.fr

Ludovic Garcia†

†EUTECH SSII
La Chapelle Saint Luc, France
{k.sokolova, l.garcia}@eutech-ssii.com

***Abstract*—Nowadays, the demand for mobile application development is high. To be competitive, a mobile application should be cost-effective and should be of good quality. The architecture choice is important to ensure the quality of the application over time and to reduce development time. Two main leaders are very represented on the mobile market: Apple (iOS) and Google (Android). The iOS development is based on the Model-View-Controller design pattern and is well structured. The Android system does not require any model: the architecture choice and the application quality highly depends on the developer experience. Heterogeneous solutions slow down the developer, while the one known design pattern could not only boost development time, but improve the maintainability, extensibility and performance of the application. In this work, we investigate widely used architectural design patterns and propose a unified architecture model adapted to Android development. We provide implementation examples and test the efficiency of the proposed architecture by implementing it on real applications.**

Keywords–Smart mobile devices (smartphones, tablets); design patterns; Model-View-Controller; Android architecture model; Fragments; Android passive MVC.

I. INTRODUCTION

This paper is an extended version of the conference proceedings [1].

The mobile market has grown rapidly in recent years. Many enterprises feel the need to be present on mobile markets and propose their services with mobile applications. Compared to computer programs, mobile applications often have limited functionalities, shorter shelf life and lower price. New applications should be developed fast to be cost-effective and updated often to keep users interested. The quality of the application should not be neglected, as mobile users are very pernickety and competition is stiff. Architecture choice remains important for mobile applications to ensure quality: mobile applications, as well as other systems, could be complex and evolve over time.

The demand for smartphone application development is high especially for the two market leaders: Apple (iOS) and Google (Android). Cross-platform solutions, such as PhoneGap, Rhodes Rhomobile and Titanium Appcelerator reduce development time, as one application is developed for several platforms [2], but have limited possibilities – often requiring native plug-ins. Cross-platform solutions also add complexity to the native code (e.g., web layer) that decreases the performance of the application. The support of non-native solutions could be abandoned. Moreover, the cross-platform solution forces having the same user interface for all platforms, while users of different platforms have different habits from native elements. The final application interface that is not common to the platform could be rejected by the user. Native solutions enable use of all the platformÕs options with better performance and lighter code enabling the creation of an application adapted to the platform, therefore developers often choose native software development kits (SDK).

The iOS SDK imposes the Model-View-Controller (MVC) design pattern for the iOS application development [3]. Android requires no particular architecture [4] – developers choose a suitable architecture for their applications that is especially difficult for less experienced developers. Complex applications that do not follow any architecture can end as a 'big ball of mud' code: incomprehensible and unmaintainable [5]. Suitable architecture can improve three non-functional requirements of software structural quality: extensibility, maintainability and performance. A defined architecture could additionally reduce the complexity of the code, simplify the documentation and facilitate collaboration work [6].

Android development books and tutorials are mostly focused on Android SDK technical details and user interface design. Only a few works have been dedicated to the Android application architecture, while the Android community identifies an architecture as an important part of successful system design and development. Developers open many discussions about suitable Android architecture on forums, blogs and groups.

In this work, we provide an overview of some widely used architectural patterns and propose an MVC-based architecture particularly adapted to the Android system. Android Passive MVC simplifies the development work giving the guidelines and solutions for common Android tasks enabling the creation of less complex, high-performance, extendable and maintainable applications.

We provide the detailed pattern description with possible implementations. We introduce several usage scenarios and propose an example of a social networking mobile application 'Tweetle' developed with our pattern. We also discuss the applicability of other presented patterns on Android development, special cases that are relevant to and the difference with

Android Passive MVC implementation.

We evaluate Android Passive MVC regarding the maintainability, extensibility and reusability with scenario-based software architecture evaluation method using the two implementations of 'Tweetle'. We also compare two implementations of 'TaskProjectManager' Android application made for a client by an experienced developer: Android Passive MVC implementation and an old implementation having no defined architecture. We conduct an experiment of long time pattern usage for real Android applications development: two developers applied Android Passive MVC for 10 months on their everyday Android projects and gave their feedback.

The remainder of the paper is organised as follows: the second section presents architectural patterns used in software development. Section 3 presents briefly the architecture used in iOS application development. Section 4 presents the Android SDK and existing difficulties in adapting one known architecture to Android. In Section 5, we propose a design pattern adapted to the Android environment - Android Passive MVC. Section 6 provides some typical cases that may arise while developing an Android application and the corresponding Android Passive MVC implementation. Section 7 describes a concrete example of a social networking application implemented using Android Passive MVC. In Section 8, we go further and provide an architecture for the core of an application. Section 9 evaluates the Android Passive MVC. Sections 10 and 11 discuss the applicability of other architecture presented in Section 2. Section 12 presents works related to mobile applications architecture and Section 13 concludes this work and presents some perspectives.

II. Architectural Design Patterns

We present five architectural design patterns in historical sequence. These patterns are widely used in desktop and web applications development. If mobile development assimilates similar design, developers moving from other systems could take advantage of their knowledge. Different components and existing variants of models are included in the description.

A. Model-View-Controller (MVC)

Presented in 1978, Model-View-Controller is the oldest design pattern and has been successfully applied for many systems since its creation [7][8][9].

The goal of this model is to separate business logic from presentation logic. The business logic modifications should not affect the presentation logic and vice versa [7]. MVC consists of three main components: *Model*, *View* and *Controller*. The *Model* represents data to be displayed on the screen. More generally, *Model* is a Domain model that contains the business logic, data to be manipulated and data access objects. The *View* is a visual component on the screen, such as a button. The *Controller* handles events from user actions and communicates with the *Model*. The *Controller* also communicates with the *View* directly if the *Model* does not need to be changed (e.g., scrolling action). The *View* and the *Controller* depend on the *Model*, but the *Model* is completely independent. The design pattern states that all *Views* should have a single *Controller*, but one *Controller* can be shared by several *Views*.

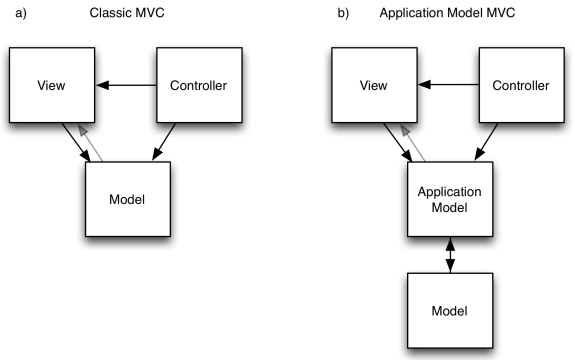


Figure 1. a) Classic MVC, b) Application Model MVC

MVC model has three varieties: Classic MVC, Passive Model MVC and Application Model MVC (AM-MVC). The scheme of Classic MVC and Application Model MVC is shown in Figure 1. The Classic MVC is shown on the left (a) and the AM-MVC is shown on the right (b). The scheme of Passive Model MVC (c) is shown in Figure 2.

In Classic MVC and Passive Model MVC, *Controller* handles events and communicates directly with a *Model* that is indicated by a black arrow. On the Classic MVC the *Model* processes data and notifies the *View*. The *View* handles messages from the *Model* and updates the screen using the data received from the *Model*. This behaviour is implemented using the Observer pattern (grey arrow in Figure 1). Conversely, the communication between the *Model* and the *View* in Passive Model MVC is done exclusively via the *Controller*. The *Model* notifies *Controller* which then notifies *View* and finally the *View* makes changes on the screen [10].

The AM-MVC is an improved Classic MVC with an additional component. The *Application Model* component was added for the presentation logic (e.g., change the screen colour if the value is greater than 4) that was often added to View or Controller previously and makes a bridge between the *Model* and the *View-Controller* couples.

B. Presentation-Abstraction-Control (PAC)

The PAC architecture was introduced in 1987 [11]. This architecture aims to improve the modularity of the system that is limited with MVC. PAC propose to decompose the system functionalities into hierarchically organised cooperating agents

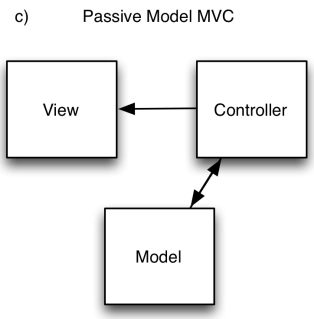


Figure 2. Passive Model MVC

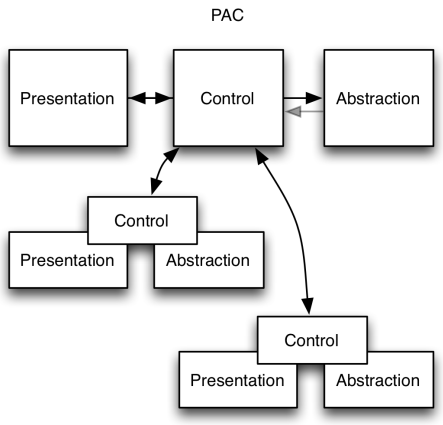


Figure 3. PAC architecture

each responsible for a particular task. Each agent manages the part of the user interface and maintains its data and state. Some agents could also exist without any particular user interface but coordinating other agents. The system can be extended by additional agents, the modification of one agent should not affect other agents.

Each agent of the PAC system consists of three components: *Presentation*, *Abstraction* and *Control*. *Presentation* component contains the presentation logic. *Abstraction* component contains the functionality of the agent and the data it maintains. *Control* component links the *Presentation* and the *Control* acting as an adapter and allows communication between agents. One can see that PAC agent is organised as Passive MVC with the difference that the user events are intercepted by the *Presentation* component [12]. Figure 3 depicts the architecture.

Agents are organised in the hierarchy where lower level agents depend on their parents. High-level agents contains the core functionalities, manage the database and main interface. Low-level agents maintain particular functionalities, particular interfaces, the information about the interface and expose actions to the user. Lower level agents could, for example, manage different sensors. Intermediate-level agents combine, maintain and coordinate low-level agents.

The actions intercepted by the low-level agents can be redirected to the upper agents to access their functionalities, the outgoing events such as an error event is also transferred to the particular 'error manager' agent via parental *Control* components. The changes in high-level agent's data are also transferred to collaborators agents.

This architecture allows a very modular system to be made with communicating agents but the system can become very complex with the fast growing number of agents. The organisation or communication between agents could also become complex.

C. Model-View-Presenter (MVP)

The Model-View-Presenter was introduced in 1996 as an MVC adaptation for the modern needs of event-driven systems [13]. The model consists of three components: *Model*, *View* and *Presenter*. In this model, the *View* represents a full screen

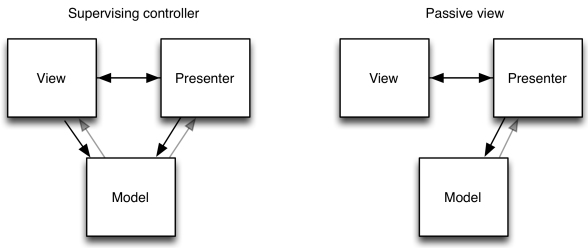


Figure 4. Supervising controller and Passive view

and it handles events from the user actions. The *Presenter* is responsible of the presentation logic. The *Model* is a Domain model.

There are two types of MVP: Supervising controller and Passive view. Both models are shown in Figure 4. The Supervising controller uses the Observer pattern for the communication between *Model* and *View*. The *View* can interact directly with the *Model* to save the data if there is no change to be made on the screen. Otherwise, the communication between the *View* and the *Model* is made via the *Presenter*. Interaction between *View* and *Model* of the Passive View MVP is done exclusively via *Presenter* [13].

D. Hierarchical-Model-View-Controller (HMVC)

The Hierarchical-Model-View-Controller was first introduced in 2000 and is similar to PAC architecture. HMVC is presented as a Classic MVC adaptation for Java programming [14]. This model takes into account the hierarchical nature of Java graphical interface components: the main window frame contains panes that contain components. The authors propose to create layered architecture for the screen with Classic MVC triads for each layer communicating with each other by *Controllers*. The HMVC model is shown in Figure 5.

Thereby the child *Controller* intercepts methods from its *View*. If a *View* of the upper hierarchy (parent *View*) needs to be changed, the child component informs the parent *Controller*, which makes the changes. The communication between layers is made exclusively via *Controllers*. Unlike PAC, the *Controllers* of HMVC have direct access to the *Model* and to core components without interacting with the high-level triad.

E. Model-View-ViewModel (MVVM)

Model-View-ViewModel is another model to separate the presentation and business logic. The *ViewModel* is a linking component between *View* and *Model*. This design pattern is mainly used in Microsoft systems [15]. The realization of this

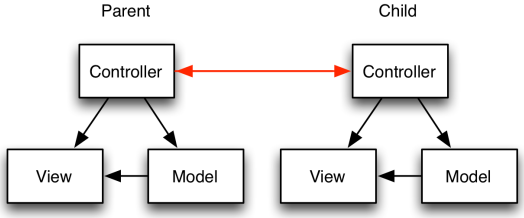


Figure 5. Hierarchical-Model-View-Controller

model is done with binding between components [16]. The binding is not supported in Android by default but could be implemented using the very recent Android-binding framework. As stated in [17], a good basic model should not use any additional framework and should be easily implemented with original components, therefore this model is not dealt with in the paper.

III. iOS APPLICATION DEVELOPMENT

The iOS mobile development has already adopted an architecture. We want to take advantage of iOS experience and knowledge in making the Android architecture. In this section we present the main principles of the architecture used in iOS development.

iOS is a OS X based system adapted to mobile devices. iOS developers use specific language called Objective C to create mobile applications.

The base architecture for mobile iOS application is an adapted Passive MVC. Like the original Passive MVC, the iOS architecture is based on three components: View, Model and Controller. Models and Views are independent and communicate with each other only via Controllers. The communication between Controllers and Model is organised via an Observer-Observable pattern.

Views and Models are highly reusable. Multiple Views are already provided by Apple: SplitView, TableView, ImageView, PageView, CellView, WebView, MapView, TextView, ButtonView, etc. Controllers are less reusable, they link Views with Models, set up the Views (contain presentation logic), and intercept actions made on View to call methods from the Model. Many controllers are already predefined in iOS: ViewController, SplitViewController, TableViewsController, etc.

A main controller for each screen or a group of screens exists in iOS applications. For example TabBar represents a menu and there are as many screens as tabs in this menu. All screens are managed by the same controller - TabBarController. Each screen can embed other Views that can have a corresponding Controller or can be managed by the parent Controller.

One can see the logic of iOS applications is similar to Android applications; knowledge of iOS architecture is helpful to adapt an Android architecture.

IV. ANDROID APPLICATION DEVELOPMENT

A. Background

Android is a Linux-based open source operation system designed for mobile devices. Android was first presented by Google in 2007 and in spite of huge competition from Apple has been the leading smartphone platform since 2010. Google continues to work on the system systematically integrating new features and correcting bugs. Many manufacturers of smartphones and tablets adopted this open-source solution; the National Security Agency (NSA) and National Aeronautics and Space Administration (NASA) also choose Android for their projects.

Android applications are mainly written in Java using the Android SDK [18]. The code is compiled to be executed on the Dalvic virtual machine on a smartphone. Additionally, developers can use the Native Development Kit (NDK) to add a C or C++ written code referred to as native. NDK allows more advanced features and better performance, however, the complexity of the code increases with the quantity of native code [19] – Google suggested minimizing the use of this kit.

Four principal components of Android SDK are used in Android application development: Activity, Service, Content provider and Broadcast receiver. Developers use predefined extendable classes to implement those components.

Activity is a main mandatory component of Android applications created when the application is opened. The simplest Android application can contain the only class implementing the Activity. Activity is also the entry point to the application: to start the application the system must launch the Activity component. Applications can make the Activity public to share the functionality it proposes.

Many Activities can exist in the application but only one is active at a time. The Activities history is saved: the system automatically maintains the stack of Activities and opens the previous Activity with its last state when the button 'back' is pressed. The oldest Activities are deleted from the stack for other memory usage.

The Service works on the background of an application permitting an execution of long tasks (e.g., file download) without freezing the screen. When the application is closed, unlike Activity, the work of the Service is not interrupted. Services can communicate directly with the Activity it is attached to.

The Content provider component gives access to the local data stored in SQLite databases. Content provider is aimed to be used for the data sharing between applications but can also be used internally.

The Broadcast receiver is a messaging system that enables communication inside the application and between multiple Android applications installed on the phone.

In 2010, Google introduces a new component into the Android systems called Fragment. Fragment is a new extendable class available in the Android SDK. Visible interface elements can now be controlled by Fragments instead of Activity, which permits the elaboration of more flexible interfaces. Therefore, part of the interface can be changed by treplacing one Fragment with another Fragment. Each Fragment is attached to the Activity and maintains access to the Activity.

Fragments main intention was to simplify the adaptability of an application between smartphones and tablets where two screens on a smartphone can become a single screen on a tablet due to the size difference. Fragments increase the modularity of the Android applications.

An exhaustive description of Android development environment and modules can be found in [20].

B. Experience

Activity causes major difficulties in implementing the known architecture: is it a *View*, a *Controller*, a *Presenter* or

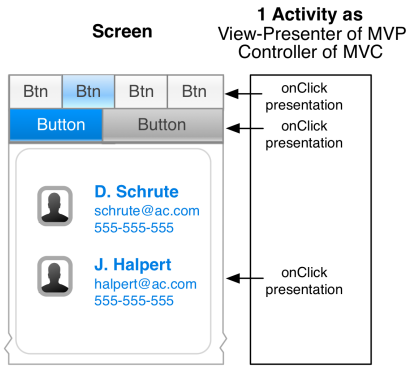


Figure 6. Activity as 'View-Presenter' of MVP or a 'Controller' of MVC

none of them?

One can observe that Android SDK already integrates many simple *Views* such as Button, TextView, ImageView, EditText and also more complex *Views* such as ListView, AdapterView, etc. One can also find several *Controllers* such as ViewFlipper, ViewSwitcher, etc. *Views* can be combined together on the screen by layout.xml and even embeds other *Views*, defining the appearance.

The most common way to develop Android application is to create one Activity per screen. Naturally, Activity initialize *Views* and intercepts actions made on *Views* by the user (methods corresponding to actions could be directly defined in layout.xml, Activity should implement the defined methods). Presentation logic of the full screen and a communication with the core of an application is often situated in the Activity making it very heavy and complex [21]. Thereby Activity managing actions and the presentation logic of the full screen behaves as a *View-Presenter* couple of MVP or a big *Controller* of MVC. The simple schema is shown in Figure 6.

We also found examples where a single Activity manages an entire application: all possible actions of an application and the full presentation logic is managed by only one class - Activity.

The *View-Presenter* or thick *Controller* implementation leads to multiple problems: reutilization, maintenance, extensibility, code clarity, team development and even performance. Parts of code integrated into single Activity cannot be reused, methods can only be copied to another Activity making the redundant code. Any additional *View* and action complicates the Activity. A modification of one action repeating on several screens requires modification in all related Activities (assuming one Activity per screen). Activity can contain the implementation of very different actions non related to each other, this can make the Activity very complex, unreadable and incomprehensible. Activity is kept in memory while the application is running, thereby a very big Activity affects performance. Finally, the modification in the user interface and application logic can lead to the need of full redevelopment of all Activities.

Some developers improve the architecture placing the Activity as a MVP *View* and put the presentation logic to the *Presenter* component. It makes Activity lighter as it manages only actions available on one screen, but reutilization and maintenance problems remain the same as explained above. The simple schema is shown in Figure 7.

Another MVC implementation place Activity instead of *View* and creates the Controller separately. Activity cannot be implemented as a *View* due to the particularity of the component, but Activity can initiate and regroup all *Views* on the screen. Thereby we obtain very thin Activity and thick Controller handling all screen events and managing the presentation logic. The simple schema is shown on Figure 8.

Even if *Controllers* could be reused by other Activities the full object is needed to reuse methods related to one View from the previous screen; the structure of application becomes unclear due to the reutilization. Problems of extensibility and maintenance persist.

This solution works for simple applications where one Activity represents one visual block, while Activity usually manages several *Views*: main screen, menu, dialogue box, lists, forms, etc. In complex visual applications *Controllers* becomes heavy.

Assuming the Activity cannot be a *View*, as *Views* are already available and extensible on Android, few developers replace the MVP *Presenter* with Activity. The simple schema is shown in Figure 9.

This solution makes *View* intercept event of all visual components available in the screen; presentation logic moves to Activity, but similar problems appear: reusability, extensibility, code clarity, etc. Presentation logic cannot be reused, but should be copied to another Activity if needed. The complexity of a single *View* increases with the number of events. This is suitable only for very simple applications with very simple screens.

The appearance of Fragments could have solved the architecture ambiguity, but Google proposes new components without suitable documentation about the utilisation of Fragments, thereby creating new ambiguity instead of solving the problem. Now developers ask themselves in what cases they should use the Fragments and not the simple Activity, what component should handle actions and presentation logic, where to place the Fragment management code, etc. We find previously explained MVC/MVP solutions, where the component that is not implemented as Activity becomes a Fragment (e.g., MVP implementation where Presenter is implemented as Activity and View is implemented as Fragment).

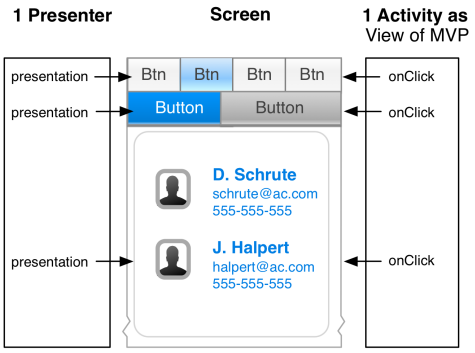


Figure 7. Activity as 'View' of MVP with additional Presenter component

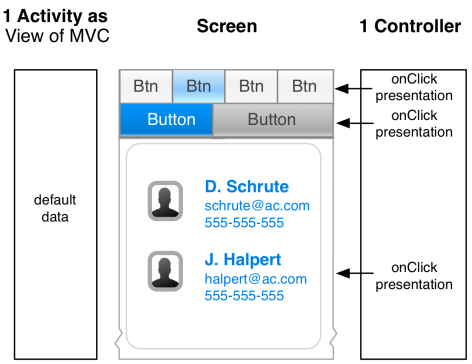


Figure 8. Activity as 'View' of MVC with additional Controller component.

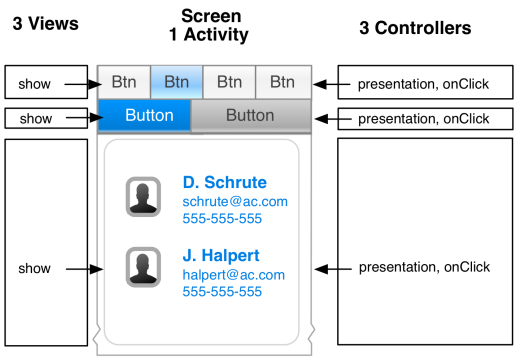


Figure 10. Activity as an intermediate component between Views and Controllers.

Nowadays more and more developers use Fragments, often as *Controllers* of MVC, but questions about presentation logic, communication between components, the actual purpose of components and its logic remains unanswered.

The full code organisation needs to be clarified: what existing component should be used and for what purpose, what type of code can be placed in those components and when and for what purpose should additional components be created? We find many applications where the part of core logic of an application is placed in the Activity or in the *Controller/Presenter* making them even more complex. Developers are often unsure about the decomposition of an application to Activities and Fragments and have problems in core organisation.

We did not find any Android application example developed using HMVC or PAC architectures. The implementations of MVVM requires additional libraries, therefore we do not take them into account.

V. Android Passive MVC : Presentation

Even if MVC and MVP architectures seem suitable for Android developments they are not intuitive to implement. The main defined problem is an Activity component that is hardly reusable. We aim to define a new architecture that can be easily implemented with Android-specific components, such as Activity and Fragments. The implementation of the model should improve the application and code quality: reduce the complexity of an application, clarify the code and improve extensibility. The coupling between components should be weak to avoid the modification of other components if one is modified. Modules should be reusable [17][22]. A mobile phone has a limited memory, therefore the creation of unnecessary objects should be avoided. Objects remaining in the memory should be lightweight [19]. Modification in user interface or in navigation logic should involve the minimum modification of the application.

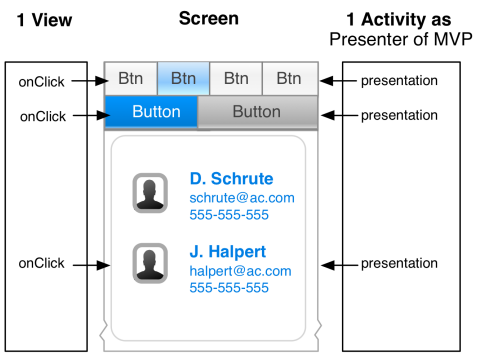


Figure 9. Activity as 'Presenter' of MVP with additional View component.

In this section we present in details our proposition: the architecture for Android application development we named Android Passive MVC.

We have decided to base our architecture on the MVC model, as MVC is well-known and widely used in desktop and web systems as well as in iOS mobile development. Developers coming from other systems would be able to easily appropriate the Android development architecture.

Activity is an inevitable component of the Android application. Previous experience of the Android community shows Activity does not fit well on the MVC model, while it seems to be well adapted to developers' needs. Many View components are already available on Android but Activity cannot be a Controller or a Model. From the previously described development experiences one can see that the screen cannot be represented entirely by one or two components. It becomes trivial that the screen should be decomposed into many logical parts and each part should have the related components. For the new architecture we decided to create MVC triads around Activity making the Activity the fourth component.

We can think of Activity as a main screen (parent) controller in HMVC model. The simple schema is shown in Figure 10.

An observer-observable pattern is relevant for multi-screen systems but only one screen is active at a time in Android applications. This pattern implies keeping in memory Views and Models that appear heavy for the mobile environment, therefore we chose the Passive Model MVC as a basis for our architecture.

In our model, Activity becomes an intermediate component between the Views and the Controllers. The Activity represents a screen controller or, in some cases, a main controller for a group of logically conjoint screens.

Controllers take the event handling responsibility and the presentation logic making the Activity lightweight. Controllers are also lightweight because one Activity can interact with many small reusable Controllers. Controller handles events and presentation logic only for a small number of views logically linked together. Controllers should not contain any code related to the core application functionalities.

The Views are the interface components, such as a form, a menu or a list of elements. View components contain methods that allow the setting or obtaining of data from the user interface on Controller demand, the setting of event listeners on visual components and the modification of visual components (set errors, change colours, etc.). Views are created if necessary extending Android predefined Views, otherwise the Android predefined Views fit to the architecture. Views are independent and do not communicate. Views should not contain any application logic or data.

The Model in our architecture is a Domain Model containing the application core logic and data. The simple scheme of the Android Passive MVC architecture with all components is shown in Figure 11.

The starting Activity creates a link between a View and a corresponding Controller to make them communicate directly. Controller set up the View it is responsible of: visual presentation and the data. The Controller handles events from the user action (e.g., button click), calls necessary methods from the Model and then updates the View on Model response.

Simple hierarchy of Activity and Controllers depending on this Activity will be suitable for many simple applications, although Android interface is a modular interface similar to Java. We propose to organise View-Controller couples in Hierarchy as HMVC and PAC architectures. The actions of interface modules controlling another interface module will be organised as parent-child controllers.

We define two type of controller: Mediate Controller and Coordinating Controllers. We borrowed names from iOS architecture also having two types of controllers.

Coordinating Controller is a simple Controller for an independent part of the screen coordinating the presentation logic and events of its Views. This Controller can call the Model, modify its View visualisation, show dialogues, call Activities but does not exchange Controllers. The Coordinating Controllers do not have any child controllers. Coordinating Controllers are very reusable and make an application very modular. Coordinating controllers can be perceived as low-level PAC agents.

Mediate Controller often corresponds to the part of the interface modifying or exchanging Coordinating Controllers (part of the interface). Menus in the interface would often

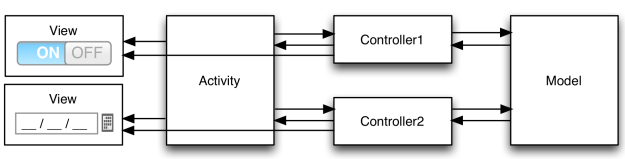


Figure 11. Android Passive MVC

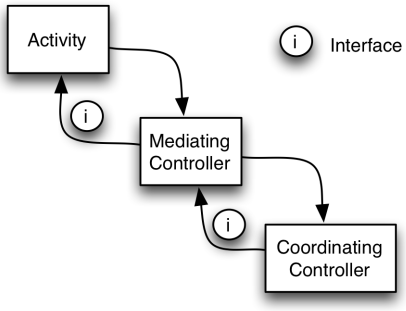


Figure 12. Communication between Controllers

correspond to the Mediate Controller. Mediate controllers can also initialise child Mediate Controllers (e.g., for a submenu). Activity Mediate Controller manages Activity replacement. Mediate controllers are similar to the Intermediate-level PAC agents with the difference that they have direct access to the Domain Model (application core).

Mediate Controllers are not very reusable as they need all their children to function, although Mediate Controllers show the presentation logic of the application; the logic of the interface can be modified by updating or changing the Mediator controller.

To keep components loosely coupled it is recommended to ensure communication between Controllers and Activity via interfaces. The communication schema is shown in Figure 12.

Android Passive MVC makes Activity lightweight by moving all event handlers and presentation logic to Controllers and interface management to Views. Views and Controllers created on demand avoid unnecessary objects, saving memory. Android predefined View fits the model and new developer Views are reusable in future applications. Coordinating controllers are very reusable and makes the application very modular. Mediate Controllers are less reusable but enable easy modification of the logic of the application only by modifying the Mediate Controllers.

Developers can easily modify or remove application components by only updating or deleting the corresponding View-Controller couple. Application can be extended with View-Controller couples. The Model is independent from the View, the Controller and the Activity. The user interface could be replaced without any impact on Model, therefore the maintainability of the application is high.

VI. Android Passive MVC: Implementation

This section presents some examples of Android Passive MVC implementation. We introduce more details and special cases of architecture usage. Controllers of AP-MVC can be implemented with simple Java classes or with the Android Fragment component.

Both implementations are suitable for the new manually created Activities. Some predefined Activities, especially from third-party libraries, will possibly not fit the implementation.

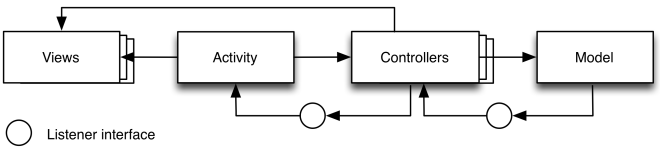


Figure 13. Android Passive MVC implementation

A. Java classes implementation

Controllers can be implemented as simple Java classes, the same as Views. Controller should be linked to the Activity, therefore the Activity should implement a Controller listener interface and pass it to the Controller to establish the communication. The components communicating via listeners are loosely coupled and the communication of Android components via interfaces is presented in [23].

As the Activity would initialise the Controller, it can communicate with Controller directly, but the communication via interface is preferable. Activity should also retrieve the View and pass this View to the Controller to establish a direct communication. We propose to establish the communication between the Controller and the Model via listeners (interfaces).

Figure 13 shows the Android Passive MVC implementation diagram. Listeners increase the performance of the application and create a weak coupling between components that improve maintainability.

For the example showing the implementation without Fragments we created a login screen with a classic login form to enter the login and password; if the login is successful the user goes to the welcome page, otherwise an error message appears.

The example contains two Activities: Login Activity managing the login page and Welcome Activity for the welcome page. The login form is managed by Login View and Login Controller. Login Activity implements the LoginControllerListener interface to be able to receive calls from the Login Controller. The schema is shown in Figure 14.

Login View contains methods for obtaining login and password (getters), methods to set button listener and methods to set errors. Login Controller handles events from the login form implementing the onClickListener; while the button is pressed, Controller calls the model that launches simple verifications. If login is successful, Controller opens a welcome screen. If login fails Controller sets up an error message.

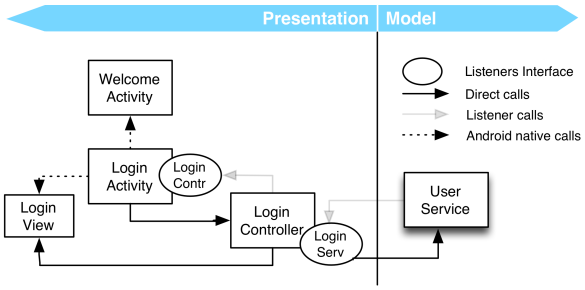


Figure 14. Login implementation example

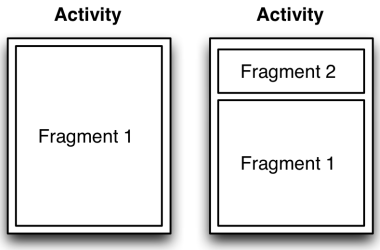


Figure 15. AP-MVC impose the creation of Fragment event if the only one is currently used within Activity

B. Fragments implementation

Fragments is an Activity-like component that can represent and control a part of the interface. Fragments can be used to implement Controllers in Android Passive MVC. Since the introduction of Fragments, Google insists on the high usage and integration of Fragments into Android applications and deprecates Activity-based functionalities.

Fragments propose multiple advantages in Android Controller implementation versus simple Java classes:

- Fragments are native Android components automatically linked to the Activity via layout.xml having the native possibility to communicate with the Activity.
- Fragments have their life cycle linked to the Activity.
- Fragments are automatically linked to Views via layout.xml and can retrieve Views to communicate directly.
- Android integrates the Fragment manager: Fragments can be easily replaced, deleted or added to the Activity.
- Activity has access to all attached Fragments.
- Fragments integrate the back button gesture: option of saving the Fragment with its state in the back stack and retrieving it on back button press. We can also choose to retrieve the existing Fragment with its state or create a new Fragment with the default state.
- Fragments can manage other Fragments.

A Fragment is created for each piece of an interface having an action or several logically linked actions. We propose to distribute all actions between Fragments and do not add user actions directly to the Activity. Even for only one simple form (e.g., login form) the Android Passive MVC imposes the use of the Controller (Fragment) along with the Activity. This makes the application more modular and improves maintainability, the same independent Fragment can be easily reused in the future. Figure 15 shows a single Activity with a single Fragment and a single Activity with two independent fragments.

Fragment is linked to corresponding Views via the layout.xml. Fragment should not retrieve other Views available in the Activity to stay independent.

Fragment can play the role of Mediate Controllers and manage other Fragments or change Activity. One Fragment cannot exchange itself with another Fragment therefore it

needs a parent Fragment (Mediate Controller) to perform the transaction. Figure 16 illustrates an example.

Android gives the option of adding a Fragment that is not directly linked to the interface, permitting the creation of Mediate Controllers without visual components. In some cases, actions from different screens and different Activities can be combined in one single Mediate Controller if those screens are logically linked. For example, a form can have several pages (screens) with 'next' button or 'go to first page' button; the appearance of the screen changes on click event interception. In this case, rather than adding an action to each fragment separately, making them dependent, the developer should create a Mediate Controller combining those actions in one place. Figure 17 illustrates this example. Therefore, in the case of user interface reorganisation (e.g., add new screen in the middle of the chain) only the Mediate Controller needs to be modified. The same should be done for a bundle of dependent fragments within a single Activity.

Fragment initialises itself with default data or the data recovered from the bundle (Android mechanism to pass the data between Activities), therefore Fragments stay maximally independent from other Fragments. Some Fragments can be initialised by an Activity or parent Fragment (Mediate Controller) to increase reusability. The possible communication between Fragments and Activities is shown on Figure 18 with two Mediate Controllers and one (the upper right) Coordinating Controller. Fragments should rarely have a callback to parent Fragments but if necessary the callback can be implemented with interfaces.

Developer should avoid high hierarchy between Fragments within a single Activity as parent Fragment is linked to the child Fragments. Activities make components more independent and simplifies the Fragment management.

In some cases, Fragments depend on each other (cannot be a parent-child, but should initialise each other), we observe the circular dependency between Fragments. Figure 19 shows an example: a list of folders and a file path to the parent folder. By clicking on the folder in the file path, the folder list should be updated; by clicking on the folder in the list, the file path should be updated. Another example is a mobile tablet with a large screen that contains the statistics data represented in different Views: tables or graphs. Changes in any of the Views should affect all other Views.

This is also a typical case where the Classic MVC is very pertinent where the Observer-Observable pattern can be used

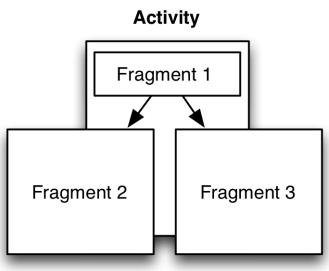


Figure 16. Mediate Fragment corresponding to the possible menu that exchange 2 Fragments depending on intercepted action

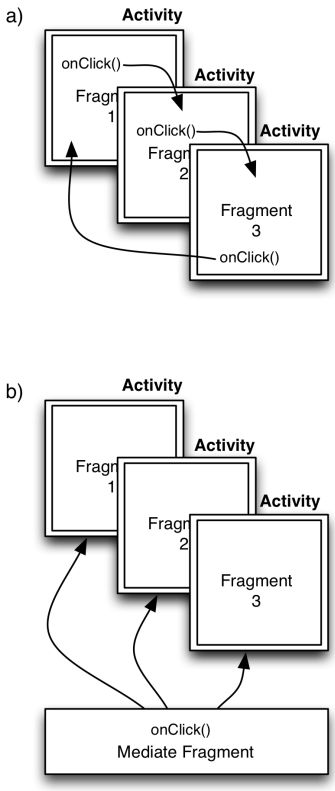


Figure 17. A chain of dependent Fragments or Activities. a) Direct calls make dependent Fragments b) Mediate Controller makes components independent

instead of Mediate Controller: several Views represent the data using the same Model, Controllers can modify the Model and all Views should be updated. Although Mediate Controller keeps components more independent.

It is possible in Android to retrieve one Fragment from another Fragment and to call the initialisation method. Although this makes very tight coupled components. A better way is to make those Fragments communicate via listeners implemented by a parent component: a Fragment playing the role of Mediating Controller.

VII. Concrete Application with Android Passive MVC: 'Tweetle'

To illustrate the implementation mechanism we take an example of a Twitter client (microblogging social network) with three buttons (main menu); one screen has an additional

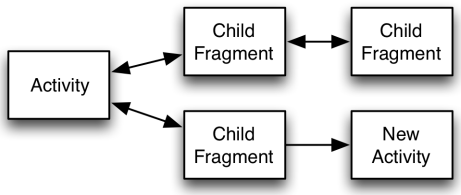


Figure 18. Communication between Fragments and Activity

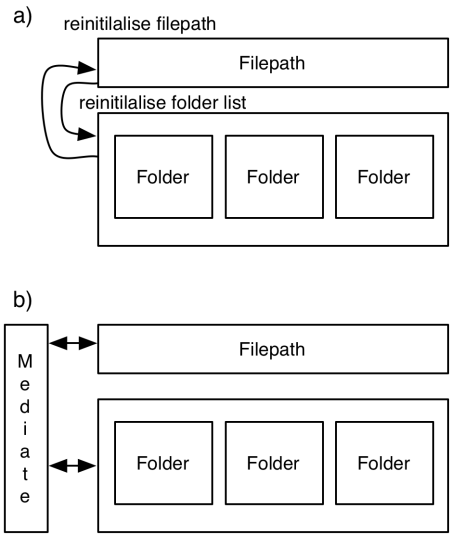


Figure 19. Circular dependency between Fragments a) Direct calls, dependent Fragments b) Mediator Controller makes Fragments independent

submenu. The user can see the Twitter timeline, send tweets with and visualise the list of tweets of his followers and followees. The bar with the copyright button showing the application authorÕs name appears permanently on the up of the screen. bar with the copyright button showing author's name of the application appears permanently on the up of the screen. The interface of 'Tweetle' is depicted in Figure 20.

One can see that all three screens are logically linked together by the main menu; one screen is divided into two logically linked parts by the submenu. Main actions are clicks on the main menu and clicks on the submenu. Additionally, by clicking on any of the list, a user can retweet the message. Finally, the button sending the tweet is presented on the last screen.

One can notice that we need two Mediate Controllers for the main menu and a submenu and at least two Coordinating Controllers for the copyright bar and the list of tweets.

Application can be implemented in two ways. We present

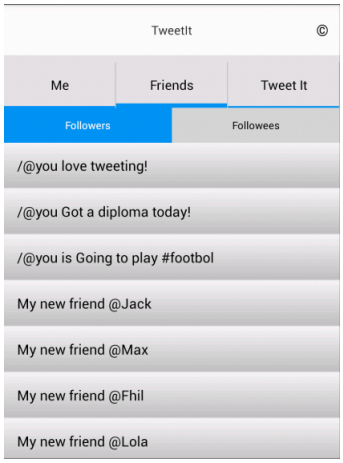


Figure 20. 'Tweetle' application user interface with followers/followees messages active tab

both implementations and discuss advantages and disadvantages of each.

1) First implementation: each different screen interface is manages by a separate Activity. This possibility is similar to "before fragments appears" implementation solutions. In our example, the main menu imposes three Activities with three buttons. One can also suppose to create one Activity by submenu or to keep the single Activity for both submenu actions as we did in our implementation example as modifications on the screen are minimal.

The screen can be decomposed into four Fragments: mostly repetitive elements on the screen. Fragment should only contain the presentation logic and actions that are logically linked together. Different actions like "retweet" and "onMenuPressed" need different Fragments.

The bar containing the copyright button corresponds to a copyright Coordinating Controller (Fragment). This controller is highly reusable even for different applications of the same developer. The bar and the button can be personalised with an layout.xml, but the Controller containing copyright action calling the dialog or a new Activity can be reused exactly in the same way in another application.

The second Fragment is a main menu Mediating Controller. This controller will change the screen (Activity) depending on the button pressed. Controller also manages the presentation of the main menu: active and non-active buttons.

The third Fragment is a list Fragment: retweet Coordinating Controller. We can reuse the same Fragment for all lists as the user action is the same for all lists of the application and there is no presentation logic.

The fourth Fragment corresponds to the submenu Mediating Controller and manages the changes in the data of the list (reinitialise the data or change the Fragment) and the presentation logic of active and non-active buttons.

Last Fragment corresponds to the form permitting to send the tweet - tweet Coordinating Controller.

Activity plays the role of an initialiser of child Controllers or a main Mediating Controller. Mediating Controllers can also initialise themselves using the data from the bundle to be more independent. Copyright Fragment is attached only by the layout.xml and do not need any additional initialisation. Activity initialise the main menu: call the Fragment method to set up active button and event listeners. Activity also initialises the list of tweets of the first screen: Activity as a main Controller can call the Model to retrieve the data and to set it to the list Coordinating Controller. List Coordinating Controller (Fragment) can retrieve the data itself either. For the Followers/Followees screen the submenu Mediate Controller (Fragment) with its default state is attached automatically to the Activity. Submenu Fragment initialises the list of tweets. Initialisation calls are depicted in Figure 21.

2) Second implementation: create an Activity for a group of logically linked screen. In our example we only have one Activity. Fragments remains the same: one submenu Fragment, one list Fragment and 'send message form' Fragment.

Menu actions can be implemented either in the Activity or a Mediator Controller (Fragment). We suggest to keep the

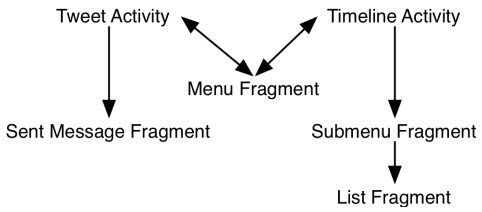


Figure 21. Activity by screen initialisation calls

main menu independent in the Fragment instead of adding it to the Activity directly to enforce maintainability. Main menu Fragment manages clicks on buttons, apply visual modifications on buttons and exchanges other visible Fragments. The submenu Fragment have an child list fragment to manage: the information shown by the list depends on the action made on the submenu. Initialisation call schema is depicted in Figure 22.

3) Both implementations: are very similar but have advantages and disadvantages.

The first implementation is easy to set up and keeps the structure clear. Activities are nearly empty thereby the only active fragments take place in memory, the number of fragments is also limited and easy to manage. 'Back' button is manages automatically. Android integrates a bundle mechanism allowing information to pass between Activities; Fragments could initialise themselves retrieving the information from the bundle while the Activity is changed. Otherwise, this implementation is only suitable for lightweight interfaces as all Fragments are reinitialised for each screen. The time response increases significantly if heavy images appear on the interface.

The second implementation has a clear structure but could be trickier to manage. This solution permits to reinitialise only necessary fragments, therefore can be used with more heavy static images, for example with the background image. Although, developer should assure to keep in memory only visible fragments. A large number of Fragments managed by a single Activity can be complicated and heavy if all Fragments are kept in memory. Fragment should be manually added to the back stack to manage the 'back' button. The second implementation is also useful for Activities aimed at being shared and at returning messages to other applications: this type of functionality should be implemented within a single Activity that another application could call for result.

VIII. Android Domain Model

The clear separation of presentation and business logic cannot ensure the application of good quality alone. The core of the application should also be implemented through patterns and gold architectures. Android application business logic structure is similar to any Java application core logic, therefore all patterns that can be applied to Java could also be applicable to the Android Domain Model, although we observe difficulties in Android Domain Model organisation.

In this section we go further and give some guidelines of the business logic of the application – the Model. Android applications have similar needs: internal database management and access, web service access and reusable components use. Clear main architecture of business logic is necessary to obtain the application of quality.

The Model of Android Passive MVC is a Domain Model containing business methods, web service call methods, database access objects, reusable methods and data model objects.

A Domain Model architecture should include components that are usual for Android applications, such as Database manager, Web services manager and Business logic. Those components should be independent, as the architecture should be adaptable. Reusable components should be also separated. The basic model architecture is shown in Figure 23.

The architecture of Domain Model proposed in this document is inspired by 3-tier architecture that separates the presentation, the business and the data access layers [24].

The business layer of our model regroups objects and methods that use web services, business services and reusable tools. Business services contain business logic. If an application works via Internet as well as locally, all necessary verifications are done in Business services, which calls corresponding methods. The communication between a presentation and a domain model layer are made via Business services.

The data layer contains Models, Data Access Objects (DAO) and Database Manager. DAO and Model are the implementation of the Data Access Object pattern. Model contains data being persisted in the database or retrieved by web services calls. Model is a simple Plain Old Java Object (POJO) that contains only variables and their getter and setter methods. To avoid transcription of the Android Native Cursor object to Model objects, Model can encapsulate the Cursor object proposing getters and setters for a concrete value type available in Cursor. Data is manipulated and transferred through the

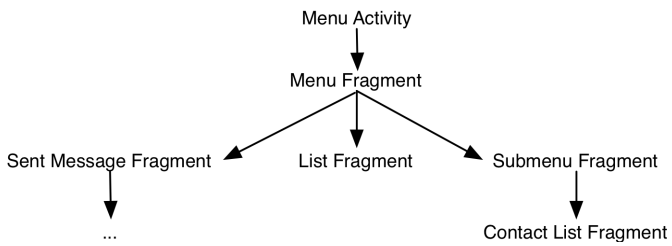


Figure 22. Activity as Main Menu

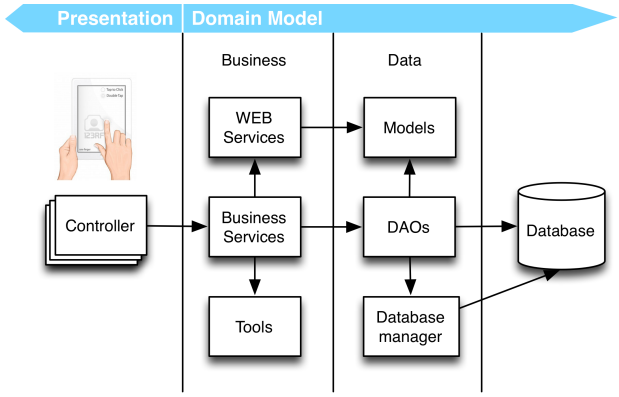


Figure 23. Domain Model Architecture

application using those lightweight objects that are often called Data Transfer Object (DTO).

Persistence methods are organized in DAOs. DAO contains methods that enable the data in a database to be saved, deleted, updated and retrieved. Even if Android proposes an abstraction on the data access level with Content Provider, DAO simplifies the code of the application. The DAO design pattern creates a weak coupling between components and uses a Model object instead of an Android Cursor object in the application. DAO can also be used for the data stored in XML or text files. Good practice is to make DAO accessible via interfaces. It allows DAO modification (for example the change of SQLite to XML storage) without any change in Business services, which increases maintainability.

Database manager is in charge of database creation. Database manager exists only if SQlite database is used by the application. It stores the name of the database, and of its tables and methods to be able to create, drop, open and close the database.

This architecture regroups logically similar methods together, increasing cohesion. High cohesion facilitates the maintainability of the software. The final code of the application could be organized in packages by architectural component: Activities, Views, Controllers, Business Services, Tools, Web Services, Model, DAOs and Database. It gives the clear structure of an application and limits the package number. Additional packages could be created for interfaces, parsers (e.g., XML, JSON) and constants.

IX. Architecture evaluation

We evaluate the architecture in two steps. First, we ensure that the architecture fits the lists of code quality criteria proposed by [17][19]. Second, we propose modification scenarios that can be applied to the 'Tweetle' and discuss the impact of each scenario on the implementation. Third, we ask an experienced Android developer to rewrite one of his latest applications using Android Passive MVC, compare results and give feedback regarding the model. Finally, we proposed to two developers that they use the architecture for 10 months in their real life projects and obtained their feedback.

A. Code quality

The evaluation of our architecture is based on the following three code quality evaluation criteria: maintainability, extensibility and reusability.

1) Maintainability: option of modifying the system.
2) Extensibility: option of adding new functionalities to the system.
3) Reusability: option of reusing the same components in different functionalities of the system or in different systems.

The use of standard platform techniques is important for the model: the support of third-party functionalities could be interrupted making implementation of the model impossible. The Android Passive MVC could be implemented using Android SDK without any additional libraries.

A high-quality application has high maintainability and extensibility: codes have weak coupling between components, easy code suppression possibility and high testability. The Passive MVC architecture ensures high maintainability. Clear separation between presentation and business logic simplifies testability of components. Weak coupling between all layers is carried out via listeners. One component (ex. interface, DAO, web service) could be replaced or modified without changes in others. The extension or modification of the user interface itself is done by simply adding, deleting or modifying the view-controller couples.

The reusability of components make the code clearer and boost development time. The view-controller components of the Android MVC model could be reused through the application and could be easily embedded in other Android applications made with Android Passive MVC.

Good performance is especially important in mobile environments: resource utilization should be limited as mobile devices have little memory. Short response time is essential for modern users. The Android MVC architecture makes a very lightweight Activity component. Controllers, View and Model objects are also small and kept in memory only if used, which minimizes resource utilization. The use of listeners also slightly increases response speed.

B. Scenario-based evaluation

We chose the scenario-based software architecture evaluation method to validate Android Passive MVC; the overview of such methods can be found in [25]. Scenarios enable evaluation of the architecture of a specific system and comparison of several architectures of the same system regarding modifiability. We apply scenario-base evaluation to previously described implementations of Android Passive MVC to show the benefits of this architecture. Most scenario-based methods involve shareholders, software designers and an evaluation team for the real project to define possible modification scenarios and the ability of the architecture to support those modifications. Our example is an illustration of the architecture, we define the most likely modification scenarios for the implementation. The two architectures of 'Tweetle' are described in Section 7.

1) Adapt the phone version to the tablet
2) Add new tab to the main menu
3) Move the main menu to the separate independent screen
4) Modify the appearance of the list
5) Add a bar containing the name of the active tab

We analyse and explain the impact of each scenario on the both implementations if different.

Table I presents the quality criteria evaluated for each scenario.

1) Scenario 1: Adapt the phone version to the tablet

'Tweetle' is an application dedicated to the smartphone usage, but can be adapted to smart tablet. Tablets in landscape mode have enough space to keep all three screens visible at one time, therefore the main menu becomes just an indicative

name menu to define each list without any action. Tablet in portrait mode will have a mobile application behaviour.

The adaptation can be easily achieved with Android Passive MVC. Application should only be adapted for the tablet landscape mode. The developer should define a new layout.xml for the new tablet appearance: the new layout mainly consists of combining the existing layouts into one. Developers do not need to define a Controller (Fragment) for the main menu as there is neither action nor presentation logic needed. Other Controllers remain the same. Developers should add to the Activity a verification of whether the tablet landscape mode is active or not and set the corresponding layout. All Controllers defined in the layout.xml will be attached automatically. One can see that a very few modifications are needed to make the adaptable interface. This scenario shows the high maintainability and reusability allowed by the Android Passive MVC.

2) Scenario 2: Add new tab to the main menu

It is very probable to add new tab to the existing menu. For example, 'Tweetle' need an extension with a map showing the newest geolocated tweets nearby. For both implementations, the developer should create a new map Fragment generating the map and Controlling actions on the map. Then, the developer can modify the main menu Fragment (controller) to add an action to the new button. For the first implementation the developer should also add a new activity-initialising fragment and an active button. Domain Model would be enriched with several new components as a new web service recovering geolocated tweets or a new DAO method recovering geolocated tweets from existing database have to be used.

One can see that only one Controller should be modified for this extension and several independent components are created. The modification of existing components is very light in Domain Model too.

3) Scenario 3: Move the main menu to the separate screen

The client wants to change the style of the mobile application creating a Windows 8-style menu screen with big square buttons and icons taking up the full screen. For the first implementation, the developer should create a new layout for the main menu and attach it to the new activity with the exact same Controller. The developer needs to check other Activities corresponding to the menu tabs to delete the initialisation of the active button, as it is not used any more if the initialisation was made in Activity.

The second implementation requires greater modifications: Activity can take a Mediator Controller role and replace the main menu with another Fragment, as the Fragment cannot replace itself. The developer could also pass to the first implementation modifying entirely the main menu Controller and creating additional Activities reusing all other fragments.

TABLE I
EVALUATION CRITERIA BY SCENARIO

Scenario #	Maintanability	Reusability	Extensibility
1	x	x	
2	x		x
3	x		
4	x		
5			x

This example shows that for maintenance reasons the developer should preferably choose different Activities for the independent screens, as in the first implementation. In spite of the common menu, all tabs are completely independent and could be arranged differently in the interface while the application evolves. Fragment Mediate Controllers are less reusable but as they are very small they can be reimplemented easily. This example shows that the architecture resists extensive visual modifications and most Controllers remain reusable.

4) Scenario 4: Modify the appearance of the list

It is possible to improve the visualisation of messages: to add an avatar, nickname, and make different colours for different lists. This can be done easily for both implementations. The developer should create an adapter to adapt the Tweet object from the Domain Model to the new visualisation in the list. The developer should only modify the adapter in the list Controller to modify the visualisation of all Controllers. If different visualisations are needed for different lists, the developer can create different Controllers or different Adapters and set up the visualisation in parent controllers.

This example also shows the maintainability of an application made with Android Passive MVC and the reusability of components.

5) Scenario 5: Add a bar containing the name of the active tab

We assume we should add a new name bar to the initial 'Tweetle' application. The visual appearance of this bar is the same for all tabs; the name corresponds to the tab name. For the first implementation, the easiest way is to add this bar directly into the layout.xml without any modification in the code. This is not possible for the second implementation. If the developer adds the modification of the view of the name bar to the main menu, two interfaces becomes dependent and cannot be used separately. Main menu could also notify the Activity to set up the name bar, but in this case the bar is not reusable in other Activities. The most reusable way to carry out the second implementation is to create a Fragment for the name bar. The main menu could pass the data to initialise the name bar as it initialises the lists instead of manipulating the view directly.

This example shows how the first implementation has maintainability advantages over the second implementation as the parent Fragments implementing Fragment transactions could be trickier to manage in case of the interface modification, but child Fragments can always be reused. This example also shows that Fragments have advantages even for the interface having no action but presentation logic.

C. Architecture application

We asked an Android developer with three years' experience to test the Android Passive MVC in real life projects. He chose to redevelop one of his latest applications which had become complex and hard to maintain, extend and test. The application is called 'TaskProjectManager' and it enables tasks to be assigned to different employees and to view the full

TABLE II
TASKPROJECTMANAGER STATISTICS

	Original	Android MVC	% Gain
# Packages	25	17	32
# Classes	393	275	30
# Functions	2186	1683	23
Avg CCN	2,30	1,87	19
Max CCN	110	30	73

calendar of tasks on the screen by day, week and month. The application also generates reports according to parameters.

We choose to compare the old version and the novel version developed using the Android Passive MVC without Fragments. In spite of the fact that developers produce slightly better code while redeveloping the same application due to greater experience, our measurements show the impact of Android Passive MVC on the redevelopment.

Measurements of both versions of the application are made with JavaNCSS, a source measurement suite for Java, and the results are shown in Table II. Android Passive MVC reduces all code parameters.

For each comparison feature denoted i, the gain is calculated as the difference between the original and the Android Passive MVC applications scores (resp. $Original_i$ and $AndroidMVC_i$) divided by the original application score (i.e., $Original_i$).

$$Gain_i = \frac{Original_i - AndroidMVC_i}{Original_i} \quad (1)$$

where:
$Original_i$ - measurement of feature i taken on the originally implemented application
$AndroidMVC_i$ - measurement of feature i taken on the Android Passive MVC implementation
i - the comparison feature

The Android Passive MVC helps with organizing classes in packages. The original version of the application had many packages created partly using the MVP model, partly the application logic, and partly the Android components names. The limited number of packages of the Android Passive MVC version gives the application a clear structure deciding the Domain Model from the interface management.

The full code became smaller: both the number of classes and the number of functions were reduced. We observed the application had a main menu appearing while the calendar was visible. Calendar had different modes of functionality managed by different activities with a huge presentation method managing the appearance of different activities. Menu actions were found multiplied in those activities. The Android Passive MVC enables high reusability of components and structuration of presentation logic.

The code complexity is evaluated using Cyclomatic Complexity Number (CCN). 'Cyclomatic complexity measures the number of linearly independent paths through a program module' [6]. Normal method complexity without any risks is 1-10 CCN, with 11-20 CCN the complexity is moderate, with 21-50 CCN the complexity is very high and with CCNs greater than 50 the program is untestable. Table II shows that the average complexity of the application has decreased slightly. The maximum CCN dropped significantly: an original version has methods with CCNs of 40, 50 and even 100 and 110 mostly for Activities handling a huge number of events, while the new version has the only JSON parser with a CCN of 30 and several methods with a CCN of 10 to 15 in the application core.

The developer's feedback explained that the Android Passive MVC model is easy to understand and to follow. The final application was visibly more reactive: the response time became almost nil, while the users of the original version complained about a very long response time for each screen. The Android Passive MVC version is open to extensions and easily modifiable. The developer said that he had already added more functionalities to the new application before transmitting the code for the CCN analysis. Application components are not only reusable in the application, but could also be reused in future Android development.

The same developer and his colleague continued to use the Android Passive MVC in their everyday job of Android application development for clients. They have tested the version with Fragments and consider it even easier due to the many predefined Android actions.

We obtained a new feedback after 10 months of testing. The developers recognise the improvement of development process since the architecture was introduced: they were able to test both types of Fragment implementations but mostly the first one. The software design state became shorter. They were able to reuse components from one application in another with slight controller modification: photo gallery, search views, 3D and PDF visualizers, horizontal scroll view, etc. They reported the shorter development time due to the clear structure defined by the architecture and easier group work. The division of projects on tasks became simpler and conflicts in code merges became less frequent.

They also noted that they were able to integrate updates rapidly while some old projects without the architecture were redeveloped entirely because of updates demanded by the client.

The developers also discovered that the architecture helps students in practical training to do better and gives them more autonomy. In older project, students without experience needed continual supervision and without it produced little maintainable code. Student, having a short practical training during the experimental Android Passive MVC usage, showed better results while having the same background as previous students involved into Android development.

The developers also noted that the architecture simplified the work with a colleague's code if he is absent, thanks to the common logic, naming and structure.

X. ANDROID AND MVP

In the MVP, the View and the Presenter correspond to a full screen interface. It is not suitable for Android, as visual block embedded into one View could be reused on several screens. Although architecture similar to MVP can be implemented

on Android around the Activity making triads for each visual piece (such as Android Passive MVC).

The Presenter manages the presentation logic of the View and communicates with the model. The difference between the implementation of both architectures is the event handling.

Events in MVP are handled by Views and the action is transmitted to the Presenter. Unlike the Android Passive MVC, in MVP View listens to the events and only calls corresponding methods in Presenter instead of letting the Controller listen to events directly. This implies the creation of additional classes for each View (visual component) implementing event listeners.

The code of Controller/Presenter remains the same. The only difference is that instead of one method handling an event (e.g., click event) Presenter uses many methods corresponding to the action to be carried out on one particular event (e.g., click on search button). This slightly reduces the Controller complexity and allows easier testing. The initial event handling method is placed in the View and remains very simple and does not need any tests.

We did not take the MVP implementation as a base architecture: the implementation of two models is very similar. The existence of Controller having a very complex event handlers can justify the use of MVP instead of Android Passive MVC but we assume it is a rare case. We assume the MVP implementation is the extension of Android Passive MVC for the exceptional particular use.

XI. Android and AM-MVC

The AM-MVC adds the Application Model (AM) component to triads moving the presentation logic to this component. We assume this component can be used in Android Passive MVC in some cases where the action of visual component remains the same but the visual presentation changes. This kind of situation is visible on 'Tweetle' implementation with reusable lists. Instead of unambiguity as to whether the population of the list should be done in the Controller or in the parent Controller, different AM components could be created for each list then, depending on the screen, Controller can initialise the necessary AM object.

We did not find this situation very common and the improvement sufficient to include this case directly into the Android Passive MVC. Similar to the MVP, we consider the AM component as a possible extension for the exceptional use.

XII. Related works

The question of mobile architecture and mobile development process was investigated since the first mobile devices, such as mobile phones and Personal Digital Assistants (PDA), appeared.

Several works were conducted on high-level (methodology) aspects studying the appropriateness of Agile methodology and proposing new methodologies for mobile application development: A Hybrid Method Engineering Approach [26], MobileD [27], etc. Among other aspects, the reusability of components was noted as a very important one.

Many projects are concentrated on web service-based mobile applications. [28] proposes the Balances MVC architecture to partition optimally the core of the application between client and server for different types of application. [29] studies the gap in mobile service-oriented application and propose a mediator layer between the mobile device and the server to fill the gap. Those authors do not include any concrete client-side architecture.

Other works were conducted on the low-level (architecture) issues. [6] conducted an experiment on the possibility of applying the Agile development on mobile systems using design patterns and proved that rapid development only benefits from defined architectures and patterns. [30] analysed the possible cases in application of MVC and PAC architectures in mobile J2ME and Symbian development and concluded that PAC is slightly more suitable due to the modularity of the interface. Authors such as [31] propose some guidelines for designing and developing mobile applications based on a single concrete implementation example.

Concerning Android systems, authors mostly concentrated on security and privacy problems rather than on application architecture. We only found the work of [23] proposing to perform a communication between all Android components via interfaces. We aimed to fill in the gap of the client-side architecture for modern Android system.

XIII. Conclusion and Future Work

The architecture plays an important role in the development of good quality applications. We identified the gap in the Android development, which was missing unified defined architecture.

We have analysed some well-known architectural design patterns and proposed an Android architecture solution based on an MVC and PAC/HMVC design pattern. We have also proposed the Domain Model organization for the Android application that helps to structure the core functionalities. We have provided implementation examples for several common cases in Android development and a concrete implementation of a Twitter client mobile application - 'Tweetle'.

The architecture defined can simplify the work of novice and experienced developers alike and enable the creation of less complex and well-structured applications.

The architecture was evaluated in several ways: scenario-based evaluation showed the high maintainability of the example implemented with Android Passive MVC. One existing Android application was reimplemented using Android Passive MVC, resulting in better maintainability, extensibility and performance. The complexity of the new implementation was considerably lower. We also involved two developers in long-term testing of the architecture on real projects and collected positive feedback on Android Passive MVC. We provided architecture explanation online to reach a larger population and to collect more feedback.

We aim to create a plug-in for Android development environments such as Eclipse or Android Studio to generate common structure, components and classes for an application. For example, the model and database classes can be generated using the database structure.

It is important to note, that Android Passive MVC could also be applied to other systems similar to HMVC and PAC architectures. It requires the main component (main Controller) implemented as Activity in Android but can be represented otherwise in another system.

REFERENCES

[1] K. Sokolova, M. Lemercier, and L. Garcia, "Android Passive MVC: a Novel Architecture Model for the Android Application Development," in Patterns 2013, IARIA, Ed., 2013, pp. 7–12.

[2] S. Allen, V. Graupera, and L. Lundrigan, Pro Smartphone Cross-Platform Development: IPhone, Blackberry, Windows Mobile and Android Development and Distribution, 1st ed. Berkely: Apress, Sep. 2010.

[3] D. Mark and J. LaMarche, More IPhone 3 Development, ser. Tackling Iphone Sdk 3. Berkely: Apress, Jan. 2010.

[4] J. Steele, N. To, S. Conder, and L. Darcey, The Android Developer's Collection. Addison-Wesley Professional, Dec. 2011.

[5] B. Foote and J. Yoder, "Big Ball of Mud," in Pattern Languages of Program Design. Addison-Wesley, 1997, pp. 653–692.

[6] T. Ihme and P. Abrahamsson, "The Use of Architectural Patterns in the Agile Software Development of Mobile Applications," in ICAM 2005 Internation Conference on Agility, Aug. 2005, pp. 155–162.

[7] G. Krasner and S. Pope, "A description of the model-view-controller user interface paradigm in the smalltalk-80 system," Journal of object oriented programming, vol. 1, 1988, pp. 26–49.

[8] P. Sauter, G. Vögler, G. Specht, and T. Flor, "A Model–View–Controller extension for pervasive multi-client user interfaces," Personal and Ubiquitous Computing, vol. 9, no. 2, Mar. 2005, pp. 100–107.

[9] M. Veit and S. Herrmann, "Model-view-controller and object teams: a perfect match of paradigms," in AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development. ACM Request Permissions, Mar. 2003, pp. 140–149.

[10] S. Burbeck. Applications Programming in Smalltalk-80TM: How to use Model-View-Controller MVC. [Online]. Available: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html (1997)

[11] J. Coutaz, "PAC," ACM SIGCHI Bulletin, vol. 19, no. 2, Oct. 1987, pp. 37–41.

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. Chichester, UK: Wiley, 1996.

[13] M. Potel, "MVP: Model-View-Presenter the taligent programming model for C++ and Java," Taligent Inc, 1996.

[14] J. Cai, R. Kapila, and G. Pal. HMVC: The layered pattern for developing strong client tiers. [Online]. Available: http://www.javaworld.com/article/2076128/design-patterns/hmvc–the-layered-pattern-for-developing-strong-client-tiers.html (2000)

[15] J. Smith, "Wpf apps with the model-view-viewmodel design pattern," MSDN magazine, Feb. 2009.

[16] R. Garofalo, Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern. Microsoft Press, Mar. 2011.

[17] S. McConnell, Tout sur le code : Pour concevoir du logiciel de qualité (Everything about code: make software of quality), 2nd ed. Dunod, Feb. 2005.

[18] R. Meier, Professional Android 4 Application Development (Wrox Professional Guides), 3rd ed. Birmingham: Wrox Press Ltd., May 2012.

[19] I. Salmre, Writing Mobile Code: Essential Software Engineering for Building Mobile Applications. Addison-Wesley Professional, Feb. 2005.

[20] S. Brahler, "Analysis of the android architecture," Karlsruher Institute of Technology, Tech. Rep., 2010.

[21] F. Garin, Android - Concevoir et développer des applications mobiles et tactiles (Android - Comprehend and develop mobile and tactile applications), 2nd ed. Dunod, Mar. 2011.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, 1st ed. Addison-Wesley Professional, Nov. 1994.

[23] W.-Y. Kim and S.-G. Park, "The 4-tier design pattern for the development of an android application," Lecture Notes in Computer Science, vol. 7105, Dec. 2011, pp. 196–203.

[24] P. D. Sheriff, Fundamentals of N-Tier Architecture, pdsa inc. ed. PDSA Inc., May 2006.

[25] M. T. Ionita, D. K. Hammer, and H. Obbink, "Scenario-based software architecture evaluation methods: An overview," in Workshop on Methods and Techniques for Software Architecture Review and Assessment at the International Conference on Software Engineering, 2002.

[26] V. Rahimian and R. Ramsin, "Designing an agile methodology for mobile software development: A hybrid method engineering approach," in Research Challenges in Information Science, 2008. RCIS 2008., 2008, pp. 337–342.

[27] P. Abrahamsson et al., "Mobile-D: an agile approach for mobile application development," in OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, Oct. 2004, pp. 174–175.

[28] H. J. La and S. D. Kim, "Balanced MVC Architecture for Developing Service-Based Mobile Applications," in e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on, 2010, pp. 292–299.

[29] A. Papageorgiou, B. Leferink, J. Eckert, N. Repp, and R. Steinmetz, "Bridging the gaps towards structured mobile SOA," in MoMM '09: Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia, Dec. 2009, pp. 288–294.

[30] D. Plakalovic and D. Simic, "Applying MVC and PAC patterns in mobile applications," Journal of Computing, vol. 2, no. 1, Jan. 2010, pp. 65–72.

[31] D. Zissis, D. Lekkas, and P. Koutsabasis, "Design and Development Guidelines for Real-Time, Geospatial Mobile Applications: Lessons from 'MarineTraffic'," in Mobile Web and Information Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 107–120.

Time-based Visualization of Large Data-Sets An Example in the Context of Automotive Engineering

Werner Sturm, René Berndt, Andreas Halm,
Torsten Ullrich, Eva Eggeling

Fraunhofer Austria Research GmbH
Visual Computing, Graz, Austria
&
Institute of ComputerGraphics and KnowledgeVisualization,
Graz University of Technology, Austria

Email: {rene.berndt, andreas.halm
torsten.ullrich, eva.eggeling}@fraunhofer.at,
werner.sturm@student.tugraz.at

Dieter W. Fellner

Institute of ComputerGraphics and KnowledgeVisualization,
Graz University of Technology, Austria
&
Fraunhofer IGD and TU Darmstadt
Darmstadt, Germany

Email: d.fellner@igd.fraunhofer.de

***Abstract*—Automotive systems can be very complex when using multiple forms of energy. To achieve better energy efficiency, engineers require specialized tools to cope with that complexity and to comprehend how energy is spread and consumed. This is especially essential to develop hybrid systems, which generate electricity by various available forms of energy. Therefore, highly specialized visualizations of multiple measured energies are needed. This paper examines several three-dimensional glyph-based visualization techniques for spatial multivariate data. Besides animated glyphs, two-dimensional visualization techniques for temporal data to allow detailed trend analysis are considered as well. Investigations revealed that Scaled Data-Driven Spheres are best suited for a detailed 3D exploration of measured data. To gain a better overview of the spatial data, Cumulative Glyphs are introduced. For trend analysis, Theme River and Stacked Area Graphs are used. All these visualization techniques are implemented as a web-based prototype without the need of additional web browser plugins using X3DOM and Data-Driven Documents.**

Keywords–scientific visualization; spatio-temporal multivariate; glyph based; trend analysis; web-based.

I. INTRODUCTION

Scientific visualization is a growing field of research in computer graphics. Recent visualization techniques have been presented, among others, at the *International Conference on Creative Content Technologies* (CONTENT), where the foundation of this article has been discussed [1].

To make fast decisions based on the data, scientists and engineers need to interpret valuable information efficiently. The complexity of data constantly increases and therefore highly specialized and individual visualization methods are needed [2].

Especially automotive engineering encounters the need to develop more efficient energy saving systems to provide longer duration with a given amount of energy. Thus, automotive engineers require specialized tools that support them to achieve this goal. An increasing research field of automotive engineering are hybrid vehicles (e.g., combination of combustion engine and electric propulsion system). As electricity is used for propulsion, several forms of energy can be used to generate electricity [3] (e.g., heat, kinetic energy or sun light). A hybrid vehicle that uses multiple forms of energy can be a very complex system. Every single component that outputs or consumes energy must be considered to ensure high efficiency. To cope with such complex systems, engineers need to be able to understand the global behavior of such hybrid systems.

Measured values of several forms of energy can be visualized to get a better understanding how energy is consumed and emitted. A measurement might be a recording of a specific scenario (e.g., start the engine and driving up the hill at 40°C temperature), thus engineers are interested in the energy trend as well. Considering that, the visualization should also represent the systems behavior over time.

In this article, several multivariate visualization techniques (two- and three dimensional) are evaluated. In addition to that, a prototypal implementation using X3DOM [4] and Data-Driven Documents (D^3) [5] technology is discussed. This interactive prototype demonstrates two glyph-based three-dimensional (3D) visualization methods including Scaled Data-Driven Spheres (SDDS) and three different two-dimensional (2D) visualization methods called ThemeRiver [6], Stacked Area Graph and Stacked Cumulative Percent Plots [7].

II. RELATED WORK

There exist many visualization techniques for multivariate data like Geometric Projection, Pixel-Oriented Techniques and Hierarchical Display [8]. The given data model is not only a general multivariate data set, but it also has temporal and spatial components, which play an important role.

Temporal or so-called time-oriented data have become a separate research field. There are various visualization techniques for temporal data and different purposes (e.g., ThemeRiver, TimeWheel) [9]. Generally, they have in common the depiction of the change over time.

On the other side, previous research also found multiple visualization methods for spatial-multivariate data, which are mostly used for scientific visualizations. These techniques include surface based rendering, direct volume rendering and glyph based techniques [10]. A large field of application for visualization methods for spatial-multivariate data are medical data visualizations (e.g., data from Nuclear Magnetic Resonance (NMR)). In the next Section, several related visualization techniques are presented, which have been considered for visualizing various forms of energy for automotive engineering.

A. Quantitative Texton Sequences

Quantitative Texton Sequences (QTonS) is a bivariate visualization technique. It maps one dimension to a spectrum color sequence and the other to a QTonS sequence (see Figure 2). These two corresponding dimensions are distributed over an 2D area. A Quantitative Texton Sequence is a set of textures (textons) as illustrated in Figure 1.

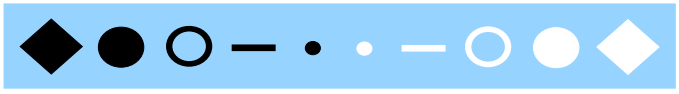
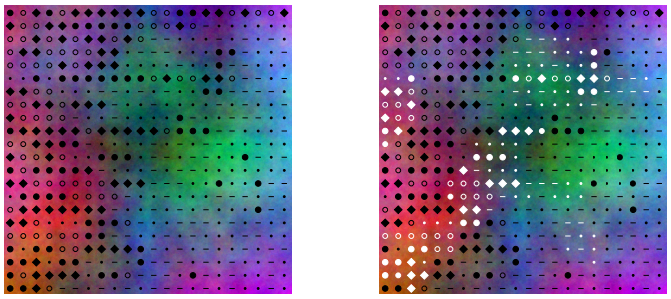


Figure 1. Example of a Quantitive Texton Sequence.

"A texton is here defined as a small texture element that when presented in a dense field forms a texture." [11].



(a) QToS with only negative values

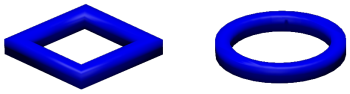
(b) QToS with both negative and positive values

Figure 2. Illustration of a Quantitive Texton Sequence visualization (QToS). Color spectrum represents the first variable and textons, placed at the same position, represent the corresponding additional variable.

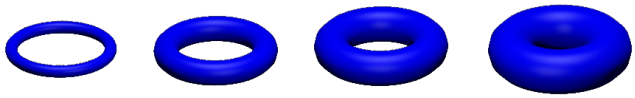
Fundamentally, values are mapped to individual Textons. The higher the absolute value is, the bigger the Textons surface will be. This means, that an area covered by Textons densily represents an area of high absolute values. To differentiate negative and positive values, black and white Textons are used. In this case, black Textons represent negative values (see Figure 2).

B. Superquadric Glyphs

Superquadrics are a category of geometrical shapes, which basically are derived from ellipsoids and quadrics. Their appearance can be changed by changing their parameters (e.g., roundness of corners, size, color, etc.). This property can be exploited to map values to the glyphs appearance.



(a) Cornered and round supertorus.



(b) Supertorus of various thickness.

Figure 3. Glyph-based visualization using superquadrics. Cornered supertorus represent negative values, round ones represent positive values (a). Absolute values are mapped to thickness of glyph (b).

Generally, this method is called glyph-based visualization and it enables a wide range of possibilities for displaying spatial multivariate data. A supertorus can be manipulated by its size (major radius), thickness (minor radius), roundness and color (as illustrated in Figure 3b and 3a). Thus, Superquadric Glyphs can be used to visualize up to four dimensions in addition to the glyphs position.

C. Scaled Data-Driven Spheres

Scaled Data-Driven Spheres is another glyph based visualization technique, which extends the 2D Data-Driven Spots [12] to 3D. SDDS use spheres as glyphs and it implies the properties of superquadrics like color and size. In addition to that, they are easier to interpret due to their simple shape and viewing angle independence. A previous user based evaluation of superquadrics and SDDS revealed that SDDS result to a lower error rate in value estimation and relationship identification [13].

Basically, SDDS use the spheres size to represent the value (see Figure 4). Scaled Data-Driven Spheres can visualize

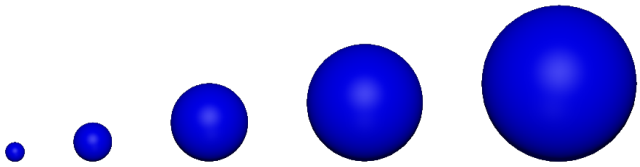


Figure 4. Glyph based visualization using Scaled Data-Driven Spheres (SDDS). Linear mapping of values to SDDS' size.

two additional dimensions (beside its position) using their properties color and size:

$$r = r_{max} \frac{v - v_{min}}{v_{max} - v_{min}}, \quad (1)$$

whereas r is the resulting radius of the sphere based on same input value $v \in [v_{min}, v_{max}]$.

D. Theme River

Theme River is a two-dimensional visualization technique designed for trend analysis over a serial dimension (e.g., time) [6], [14]. Theme River enables users to compare different identities, which occur parallel. Every identity is represented by a colored current within the river (see Figure 5).

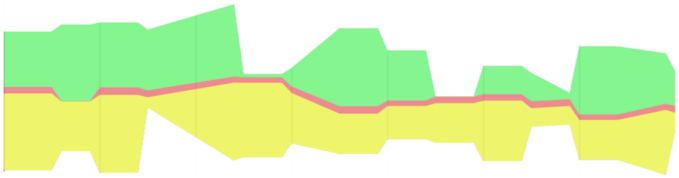


Figure 5. Illustration of Theme River with three currents. The width of current represents the corresponding value. It is primarily used to identify trends. Thus, labels indicating the exact value are not essential.

The higher a value is, the wider the corresponding current will be. If a value changes along the serial dimension, the currents width will proportionally change to it. Hence, abrupt changes and long term trends can be recognized easily. If a value equals zero, the appropriate current will disappear until it changes its value again.

E. Stacked Area Graph

Stacked Area Graph represents serial values as areas. The x-axis represents a serial dimension and the y-axis represents the corresponding value. When values change over a serial dimension, the height of the corresponding area will change as well. If multiple values are depicted, areas are not overlapping. They are arranged as a stack. Thus, the total sum of all depicted values is represented by the total hight of the stack (see Figure 6).

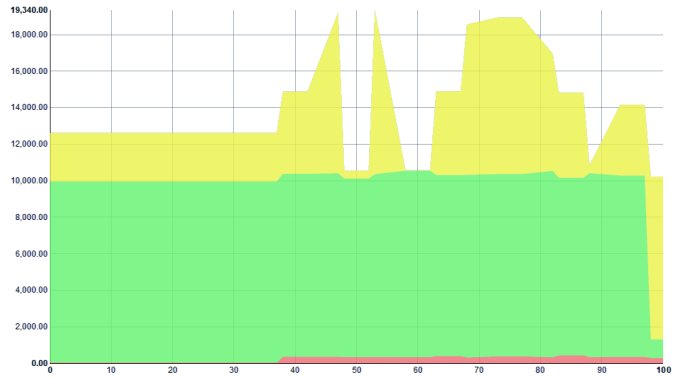


Figure 6. Illustration of Stacked Area Graph with three elements. The height of stack element expresses the corresponding value.

This enables the user to recognize the total amount easily. Like Theme River - if a value changes to zero, the corresponding area will disappear. If the value changes again, the area will appear at the same stack level where it disappeared before.

F. Stacked Cumulative Percent Plot

Stacked Cumulative Percent Plot (SCPP) is another graphical method for visualizing trends [7]. In contrast to Theme Rivers and Stacked Area Graphs, SCPP depicts the percentage instead of the value as stacked areas.

Figure 7. Illustration of Stacked Cumulative Percent Plot with three elements. The height of stack element represents the values percentage of total sum.

The total graph is always stacked up to 100% (as in Figure 7). When values are normalized, it enables the user to identify how much a single component contributes to the total amount.

G. X3DOM

As these visualization techniques have to be implemented with the aid of current web technologies, X3DOM is used to render the 3D visualization. Generally, X3DOM is a new approach to enable web browsers to render 3D content without the need of further plugins [4]. The current HTML5 standard defines X3D (derived from VRML97) to be used to provide 3D content, but it does not define how this 3D content has to be handled. For that, X3DOM has been approached to connect the web browsers *Document Object Model* (DOM) containing the X3D data, with the internal X3D runtime. The X3D runtime does all 3D tasks, which are needed to render the scene defined by the X3D data. X3DOM enables web developers to create 3D content in a declarative way as used for creating SVG and common HTML elements. Thus, there is no need to struggle with low-level graphics interfaces like OpenGL provided by WebGL [15]. Due to its observer architecture, it also supports dynamic DOM changes, which effect an analogous update of the X3DOM runtime content. X3DOM has not been declared as a web standard, but like XML3D [16], it runs for it.

H. JQuery

JQuery is a JavaScript framework, which simplifies the development of dynamic web pages [17]. It provides functionalities to modify the web browsers DOM and to invoke remote procedures via AJAX. It's *selector* enables programmers to select DOM elements by their CSS attributes and update them via implicit iterations. JQuery has become a first choice for web developers.

I. Data-Driven Documents

Data-Driven Documents (D^3) is a JavaScript library for manipulating the DOM based on data (data-driven) [5]. It is designed to cope with large data sets and it allows to bind data to the DOM combined with dynamic properties for

animations and user interactions. It uses several well known web technologies like SVG, CSS3 and HTML5 to generate a representation of the defined data. Therefore, no further libraries or plugins are needed. Moreover, it uses similar methods called selectors as JQuery does.

III. Preconditions

A. Data model

Problem-specific visualizations are tailored for specific data. Thus, it is important to know the underlying data structure. In our case, we have spatio-temporal multivariate data. The data consist of the energy type, the 3D position where the data were collected, the data acquisition time and the measured value itself. The value can be positive or negative. A single measurement represents a period and consists of various value sequences. As illustrated in Figure 8,

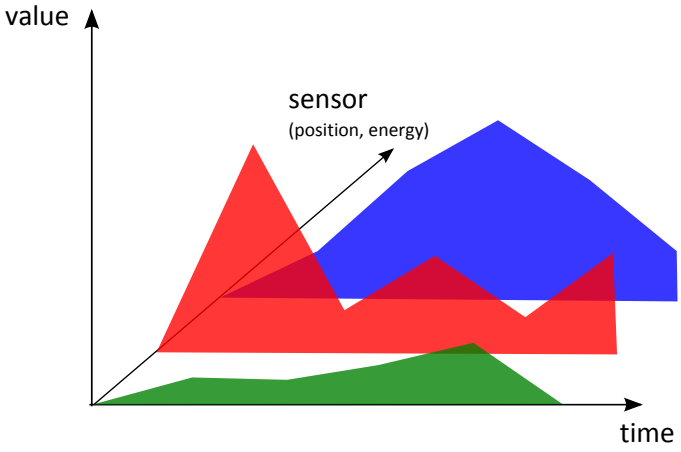


Figure 8. Representation of the data model. Dimension *sensor* combines 3D position and form of energy.

a single value sequence represents the recording of a single sensor (measuring point). All values are measured parallel and it can be assumed that all sensors have been synchronized. Hence, the n^{th} value of every value array represents the same time.

The multivariate data have six dimensions:

- form of energy
- value
- time
- 3D position (one scalar for each dimension)

B. Visualization

The visualization must be capable to display five distinct forms of energy and an arbitrary number of sensors in an explorable way. These forms of energy should be easy to interpret, to distinguish and to compare. In addition, the trend over time must be easy to investigate to satisfy the requirements of automotive engineers. To achieve that, both 2D and 3D visualization techniques will be considered.

Summing up, an optimal visualization fulfills these requirements:

- displaying at least five distinct forms of energy measured by several hundred sensors
- easy exploration of data, especially considering spatial and temporal dimensions
- opportunities to compare measuring points and different forms of energy
- gives a global overview about the data
- opportunities for detailed trend analysis

IV. Examination

Basically, the human's perception capabilities are limited to a 3-dimensional spatial sense plus a perception of time. This results to a 4D world, in which we live in. This limitation also narrows the possibilities of beneficial visualization methods, while the perception of time can only be used via animations. Thus, a single image can not exploit the perception of time. Regarding this fact, properties of geometrical shapes (glyphs) (e.g., color, size, roundness, pattern and orientation) can be used to gain the feasibility to visualize even more dimensions. This is needed to display multi-dimensional data.

An investigation of existing techniques revealed that a combination of 2D- and 3D visualization techniques lead to a better global insight into the measuring data. Moreover, this allows the user to investigate the data from different points of interest.

A. 3D visualization

If the data set has a spatial component and it is important to know its location, it is common to use a visualization technique that retains this spatial information. The drawback of this decision is, that every visualized dimension consumes a single dimension in the visualization space. For our problem, all three dimensions of the spatial variable are used because the cognition of the energy's location is essential. Therefore, no spatial dimension for visualizing time is left. Three-dimensional visualizations also allow more freedom to explore the spatial data in a natural way.

1) Quantitative Texton Sequences: QTonS are considered to be used as a render mode of the three-dimensional space to represent the two dimensions *value* and *form of energy*. While exploring the 3D-space, the view is updated by the QTonS technique to represent the amount of energy in the viewed region. The mapped colors represent the *form of energy* and the texons represent the *value* of the data (see Figure 2). The color of textons indicates the sign of its *value*. White textons represent positive values and black textons represent negative values. This combination of 3D and QTons visualizes all needed information of the measurement at a single point of time. Therefore, a control to navigate through the measuring period is needed.

The drawback of this approach is that the location of the visualized energies is not clearly evident. The user can only perceive how much energy occurred in a specific direction

calculated from the viewpoint. Therefore, it is not possible to investigate a single location. This results to an ineffective explorable visualization. Moreover, it is rather suitable for data sets of high spatial density. Such a high density might be reached by some measurements but this is not the common case. It is assumed that common measurements have up to several hundred measuring points distributed over the vehicle.

2) Superquadric Glyphs: Here, the approach called torus-based Superquadric Glpyh (supertorus) is examined, which is discussed in [18]. Compared to QTonS', this technique is more meaningful for displaying 3D-spatial data, because glyphs represent the spatial information by their 3D-position implicitly. The dimension *form of energy* is mapped to the glyphs color and the corresponding absolute *value* is mapped to its thickness as shown in Figure 3b. To indicate if a *value* is either positive or negative the roundness is used therefore (see Figure 3a). A round supertorus represents a positive value.

Superquadric Glyphs scale very well from few data sets to several hundred. The user can easily identify regions with high occurrences of a specific form of energy. If a scale is provided, the user can estimate the value of a single glyph. Therefore, glyphs can also be compared to understand relationships of different energies. To preserve the size of glyphs perspective-independent, an orthogonal projection is used. Contrasting colors should be chosen as discussed in [19] and [20] to be distinguishable easily.

A serious disadvantage of superquadrics is that users can not easily interpret the glyphs roundness and thickness when glyphs are placed densely. In addition to that, superquadrics are not viewing angle independent. According to the user study [13], Superquadric Glyphs are not as effective as Scaled Data-Driven Spheres (SDDS). The next Section will discuss SDDS in detail.

3) Scaled Data-Driven Spheres: In contrast to Superquadric Glyphs, Scaled Data-Driven Spheres are viewing angle-independent. To distinguish different types of spheres, contrasting colors are used. According to [20], humans perception struggles to differentiate more than 12 color values. Therefore, up to 11 (plus one for background) colors can be used to represent different glyphs. In this case, up to 5 different forms of energy are needed to be displayed. This allows to use 2 colors for each form of energy to distinguish positive and negative values (10 colors). To clarify that these two colors belong to the same form of energy they should have the same ground color (e.g., light red and dark red). Therefore, light colored spheres represent positive values and negative values are represented by dark colored spheres (as in Figure 9). The contrasting base colors of all five *forms of energy* are green, yellow, red, blue and magenta.

The remaining free color value can be used to display the outline of a car as a semitransparent model. This assists the user to interpret 3D positions easier. The car model should not disturb the perception of the visualized measuring data. Thus, light gray is used since gray is not a signaling color. As a result, all dimensions of the data at a specific moment can be visualized in a comprehensible way.

However, SDDS underly several side effects of the 3D rendering process. To achieve a three-dimensional impression, objects have to be shaded, which effects a distortion of the objects original color. When using a semitransparent car model, the color of the car model also distorts the color of SDDS that are located behind the model. This can result to an unclear color identification and should be considered.

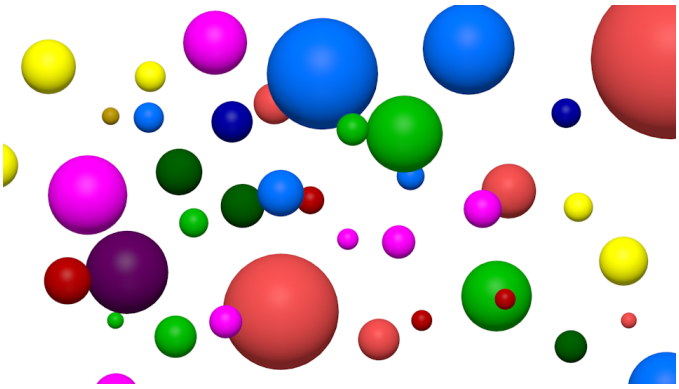


Figure 9. Illustration of cluttered Scaled Data-Driven Spheres (SDDS) using contrasting colors. Each base color represents a distinct form of energy.

To illustrate the trend over time, a key frame animation showing the change of measuring values is used. Every key frame equals one measuring time, which is calculated by mapping values to the corresponding spheres' size. The glyphs size and color (when value is changing its sign) has to be interpolated between each frame. Consequently, when key frames are shown in sequence, an animation depicting growing and shrinking spheres will be the result.

Due to the ease of interpreting SDDS, their low error rate in value estimation and relationship identification, SDDS are the most suitable visualization technique to display detailed measuring data spatially.

B. Cumulative Glyph

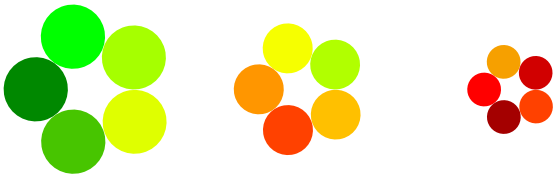
In some cases, engineers do not want to get a detailed insight in the data instantly. They might want to get a more general overview of the data to spot interesting regions to investigate more in detail afterwards. Cumulative Glyph (CG) is an approach for this demand. A CG is a representative of a specific spatial region (e.g., single wheel, engine, etc.) of the data. In a single region, there might be plenty of sensors. To understand the overall incidence of energy in this region, it still can be difficult to estimate when using SDDS. Therefore, all values of the same form of energy within this region will be cumulated to a single value. This cumulated value represents the total occurrence, which can be compared with other cumulated values of other regions.

Figure 10. Color scale used for color mapping. Green indicates high occurrence of energy. Red indicates low energy.

$$
\begin{aligned}
ratio &= \frac{v - v_{min}}{v_{max} - v_{min}} \\
red &= \begin{cases} 0.4 + \frac{v - v_{min}}{\frac{1}{4}(v_{max} - v_{min})} \cdot 0.6 & \text{if } 0 \leq ratio \leq 0.25 \\ 1 & \text{if } 0.25 < ratio \leq 0.5 \\ 1 - \frac{v - \frac{1}{2}(v_{max} + v_{min})}{\frac{1}{4}(v_{max} - v_{min})} & \text{if } 0.5 < ratio \leq 0.75 \\ 0 & \text{if } 0.75 < ratio \end{cases} \\
green &= \begin{cases} 0 & \text{if } 0 \leq ratio \leq 0.25 \\ \frac{v - \frac{1}{4}v_{max} + \frac{3}{4}v_{min}}{\frac{1}{4}(v_{max} - v_{min})} & \text{if } 0.25 < ratio \leq 0.5 \\ 1 & \text{if } 0.5 < ratio \leq 0.75 \\ 1 - \frac{v - \frac{3}{4}v_{max} - \frac{1}{4}v_{min}}{\frac{1}{4}(v_{max} - v_{min})} \cdot 0.6 & \text{if } 0.75 < ratio \end{cases} \\
blue &= 0
\end{aligned}
\tag{2}
$$

To visualize cumulated values, color mapping is used. As illustrated in Figure 10, colors as green, yellow and red are used to depict the amount of energy. Green represents a high, yellow an average and red a low (negative) amount of energy.

Despite the fact, that many color values, which are close to each other on this color scale, can not be compared easily (subjective colors), this representation has enough potential to offer the user an overview to know where high occurrences of energy exist. This color scale is chosen because the rainbow color map is commonly accepted by users. Of course, other color scales can be used that are considered to be easier to differentiate. This topic is discussed in [21], [22], [23], [24]. Every CG consists of n (where n is the number of different energies) discs, which are arranged circularly. Every disc represents a single form of energy. In this case, a CG consists of five discs. Every disc is colored by the mapping rule as Equation 2 depicts. If a form of energy has not been measured in this region, the corresponding disc will be gray and represent no value. In addition to that, the overall amount of energy



(a) QToS with only negative values

(b) QToS with both negative and positive values

(c) QToS with both negative and positive values

Figure 11. Illustration of Cumulative Glyph (CG) representing cumulative values of regions within the data space. CG of high overall cumulative value (a). CG representing a region of average cumulative value (b). CG of low energy value (c).

within a region is mapped to the CG's size. This allows the user to identify if a region involves a high occurrence of energy. Like SDDS, Cumulative Glyphs can be animated to show the energy trend over time as well. Colors of discs and the glyphs size have to be interpolated between every key frame. In this way, the user is able to get an idea how energy behaves in several locations over time.

Due to the Cumulative Glyphs shape, CG's should have a camera-relative orientation to be view-independent. As depicted in Figure 11, it is difficult to recognize what form of energy is represented by a disc. For that, it is recommended to use icons to label discs to improve the ease of interpretation or at least, a legend should be provided.

C. 2D visualization

1) Theme River: Due to Theme River's properties, it is suitable to depict the trend of measured values. While the serial dimension is the time component of a measurement, every sensor is represented by one current within the river. Currents that represent sensors measuring the same form of energy are of the same basic color and are placed side by side. As a result, these sensors together generate a wide current representing the total amount of one form of energy. Additionally, the total width of the river describes the total amount of energy. This allows users to compare trends of different forms of energies and various sensors at once.

A drawback of Theme River is the difficulty of estimating values. Without any labels, users can hardly guess what value is represented by a given width. Moreover, Theme River works only with positive values. Thus, either absolute values should be used (where negative values are represented by darker colors) or two graphs, one for positive and the other one for negative values might be a solution. The latter solution should be preferred, because the former one treats positive and negative equally, which leads to a wrong representation of the total energy occurrence.

2) Stacked Area Graph: Stacked Area Graphs can also be used to examine the energies trend over time. Every measuring point is represented by an area. As implemented in Theme River, measuring points representing the same *form of energy* are placed next to each other. In addition, they are of the same base color to represent the overall occurrence of that specific *form of energy* as a cumulated area. This enables users to investigate the trend and to identify the entire amount of energy in an area (e.g., engine or front wheel). Negative values are represented by stacks below the x-axis. To navigate through time, controls for setting the displayed interval should be provided. Possibilities to set start time and end time independently allows users to *zoom* within the dimension of time.

3) Stacked Cumulative Percent Plot: Stacked Cumulative Percent Plots are very similar to Stacked Area Graphs. Due to the depiction of percentages instead of values, users can easily recognize which energy contributes the most. To compare different *forms of energy* based on meaningful percentages, values are normalized. Furthermore, absolute values are used to handle negative values. To identify that an area represents negative values, darker colors are used (as used for SDDS).

V. PROTOTYPE

The aim of this prototype is to test all chosen visualization techniques and proofing that these techniques can be implemented as a web-enabled application using X3DOM of version 1.3 and Data-Driven Documents (version 2.9.6).

A. Architecture

As depicted in Figure 12, the main parts of the prototype are the database, the web server and the client, which is a web

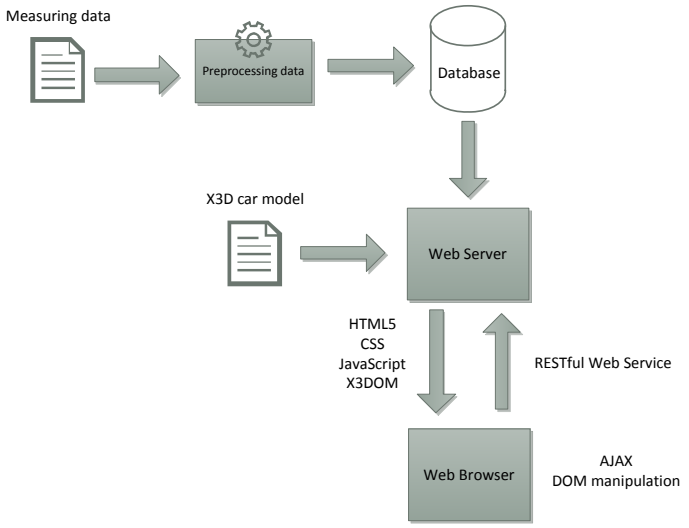


Figure 12. General architecture of prototypal implementation. Original and reduced approximated values are stored in database. Web server provides HTML content and data to the client. Client performs complete visualization using X3DOM and Data-Driven Documents.

browser supporting 3D within web sites. The major part of the applications logic is run within the clients web browser to provide dynamic visualizations, which respond quickly to user interactions. The web browser requests needed measuring data from the web server via RESTful web services and maps it into a data format that can be visualized by X3DOM or D^3. Hence, the web server primary has to provide the requested data and further visualization-specific calculations are performed client-side.

1) Data Preprocessing: Transferring all data to the client would be very time-consuming. Thus, the data are preprocessed to reduce the amount of data. Generally, a measurement with n measuring points and t value acquisitions results to a data complexity of $S(n,t) = \mathcal{O}(n \cdot t)$. As the amount of measuring data can be massive, the Douglas-Peucker algorithm is used to to reduce the number of values [25]. This algorithm has an expected complexity of $\Theta(nt \cdot \log(nt))$; worst case is $\mathcal{O}((nt)^2)$. This works because the sequence of measured values can be interpreted as a curve representing the trend (see Figure 8). After applying this geometric algorithm, an approximation of the original curve with less points, which still preserves the overall structure of the curve will be the result. In other words, this algorithm filters all non-distinctive values. Consequently, the less data the server has to find and transfer to the client, the faster the system responds to the user. Moreover, users want to navigate through the data using different time-zoom levels. For that, different degrees of approximations are needed, which are stored in the servers database after the preprocessing task.

2) Web Server: Basically, the web server provides HTML content including the static 3D car model in X3D format, which can be understood by X3DOM. However, the servers main task is to provide measuring data to the client. For that, it offers several RESTful web services, which provide the data formatted using *JavaScript Object Notation* (JSON). Of course, clients do not receive all data at once. For both 3D visualizations (SDDS and Cumulative Glyph) the client commonly requests only data of a single moment. But all time-oriented two-dimensional visualizations and animations of SDDS and Cumulative Glyphs, used for depicting the trend over time, need a data set of the desired time span. Users might select a long period (or even the complete period), which would cause a huge amount of data to be transfered to the client. Considering that, the web server limits the maximum amount of values to be transfered and selects the best approximation of the data (as discussed in Section V-A1), which does not contain more values as limited. This allows the client to request any time span without overloading.

3) Client: As already mentioned, the most of the applications logic is performed client-side by JavaScript. Users should be able to explore data in an interactive way. It uses several libraries to handle these tasks. JQuery is used for user interface interactions, performing *Asynchronous JavaScript and XML* (AJAX) calls and DOM manipulations. X3DOM is applied to visualize 3D content. The 3D scene is manipulated via DOM manipulations and D^3 provides all functionalities to draw 2D graphs.

B. 3D using X3DOM

The observer architecture of X3DOM, which connects the web browsers DOM with the internal X3D runtime, allows to modify the 3D scene via the DOM itself. Moreover, X3D predefines simple objects like box, sphere, cone and cylinder. Therefore, a SDDS can be created easily by using a sphere object (see Listing 1). Resulting X3D tags are depicted in Listing 2.

```
function showSDDS(x, y, z, size, color, id) {
  //
  // create tag <transform>
  // and append it to scene tag
  //
  $('<transform></transform>', {
        id: 'sdds' + id,
        'class': 'sdds',
        translation: x + " " + y + " " + z,
        scale: size + " " + size + " " + " " + size
    }).appendTo("#scene");
  //
  // create tag <sphere>
  // and append it to tag <shape>
  //
  $('<sphere></sphere>', {
        'class': 'sphere'
    }).appendTo(
      // create tag <shape>
      // and append it to tag <transform>
      $('<shape></shape>', {
          id: 'sphere' + id
      }).appendTo("#sdds" + id)
    );
  //
  // create tag <material>
  // and append it to tag <appearance>
  //
  $('<material></material>', {
      id: 'material' + id,
      transparency: 0.2,
      diffuseColor: color
    }).appendTo(
        // create tag <appearance>
        // and append it to tag <shape>
        $('<appearance></appearance>')
        .appendTo("#sphere" + id)
    );
}
```

Listing 1. JavaScript code to create a sphere with X3DOM representing a Scaled Data-Driven Sphere. All X3D state changes are performed via DOM manipulations using JQuery.

```
<X3D xmlns="http://www.web3d.org/specifications/x3d-
     namespace" id="x3d_element" showStat="true" showLog="
     false">
  <scene id="scene" pickMode="box">
  ...
    <transform id="sdds1" class="sdds"
        translation="0 0 0"
        scale="10 10 10">
      <shape id="sphere1">
        <sphere class="sphere"></sphere>
        <appearance>
          <material id="material1"
             transparency="0.2"
             diffusecolor="#FF6666">
          </material>
        </appearance>
      </shape>
    </transform>
  ...
  </scene>
</X3D>
```

Listing 2. X3D tags defining a Scaled Data-Driven Sphere.

Cumulative Glyphs can be created analogously using cylinders arranged circularly as depicted in Figure 13. To include a static car model provided as a file, the inline function can be used as shown in Listing 3.

```
<transform rotation="0 1 0 1.57">
  <appearance>
      <material transparency=".8"></material>
  </appearance>
  <inline id="carModel" nameSpaceName="carmodel"
     DEF="carModel" url="data/x3d/carModel.x3d"></inline>
</transform>
```

Listing 3. Include X3D car model via inline tag.

To animate glyphs to depict the values change over time, X3DOM implements *TimeSensors*, *Interpolators* and *Routes* defined by the X3D standard. *TimeSensors* provide a timer to trigger steady events within a defined period. *PositionInterpolators* are connected to *TimeSensors* via *Routes*, which provide a 3D vector interpolated between two defined vectors depending on the timers progress. These output vectors are passed to the spheres attribute *scale* (which expects a 3D vector) resulting a smooth animation showing a growing or shrinking sphere. Colors of glyphs also have to be changed during an animation. For that, X3D offers *ColorInterpolators*, which work analogously. This concept also supports long lists of key frames and therefore, any animation can be defined (see Listing 4).

1) Compatibility Issues: Nevertheless, there have been some issues with X3DOM. As mentioned in Section IV-A2, when using glyphs, an orthogonal projection should be used. Unfortunately, X3DOM does not implement *OrthoViewpoint* (defined by X3D). Thus, a perspective viewpoint is used instead. As depicted in Figures 14 and 13, transparency is applied to the car model and most glyphs are located within it, which led to several rendering problems. Many glyphs have not been rendered at all. To solve this problem, a work-around that reverses the rendering order in a way that glyphs are rendered at last is used. In addition to that, it was not possible to trigger mouse events of glyphs because there is no way to click *through* the transparent car model. Another problem was to orientate Cumulative Glyphs relatively to the camera. For that, X3D defines a *ProximitySensor* but X3DOM does not support it (yet).

```
<X3D xmlns="http://www.web3d.org/specifications/x3d-
     namespace"
    id="x3d_element" showStat="true" showLog="false">
  <scene id="scene" pickMode="box">
  ...
    <timesensor id="Timer" loop="loop"
      cycleinterval="3" enabled="true"/>
    <positioninterpolator id="PosInterpolator"
      key="0 0.5 1" keyvalue="5 15 5"/>
    <route fromnode="Timer" fromfield="fraction_changed"
      tonode="PosInterpolator" tofield="set_fraction"/>
    <route fromnode="PosInterpolator" fromfield="
        value_changed"
      tonode="sdds1" tofield="set_scale"/>
    <colorinterpolator id="ColorInterpolator"
      key="0 0.5 1"
      keyvalue="1 1 1 0 0 0 1 1 1"/>
    <route fromnode="Timer" fromfield="fraction_changed"
      tonode="ColorInterpolator" tofield="set_fraction"/>
    <route fromnode="ColorInterpolator" fromfield="
        value_changed"
      tonode="material1" tofield="diffuseColor"/>
  ...
  </scene>
</X3D>
```

Listing 4. Example of an animation with three key frames defined in X3D.

Regarding web browser compatibility, WebKit-based web browsers such as Google Chrome (version 23.0.1271.95) and Firefox (version 17.0.1) perform well with X3DOM. They also implement WebGL natively. Despite that X3DOM implements a fallback model in case that the web browser does not support 3D natively or via X3D/SAI plugin, as a last resort it tries to use Adobe's Flash plugin to render the scene. For example, this happens when using Microsoft's Internet Explorer (version 9) with no other plugins installed. Besides performance issues, the result was not acceptable because there have been to many rendering problems. This is a little insight into web browser compatibility when using X3DOM besides that evaluating compatibilities is not a goal of this paper.

C. 2D using Data-Driven Documents

An investigation of multiple graphing libraries such as Grafico [26], which is based on Raphaël and Prototype.js revealed that D^3 performs very well even with bigger data sets (e.g., 1000 data points).

```
$('#chart').append('<svg/>');
   nv.addGraph(function () {
       var chart = nv.models.stackedAreaChart()
                .x(function (d) { return d[0] })
                .y(function (d) { return d[1] })
                .clipEdge(true);
       chart.xAxis
          .tickFormat(function (d) { return d });
       chart.yAxis
          .tickFormat(d3.format(',.2f'));
       d3.select('#chart svg')
           .datum(data)
           .transition().duration(500).call(chart);
       nv.utils.windowResize(chart.update);
       return chart;
   });
```

Listing 5. JavaScript code to create a Stacked Area Graph using Data-Driven Documents.

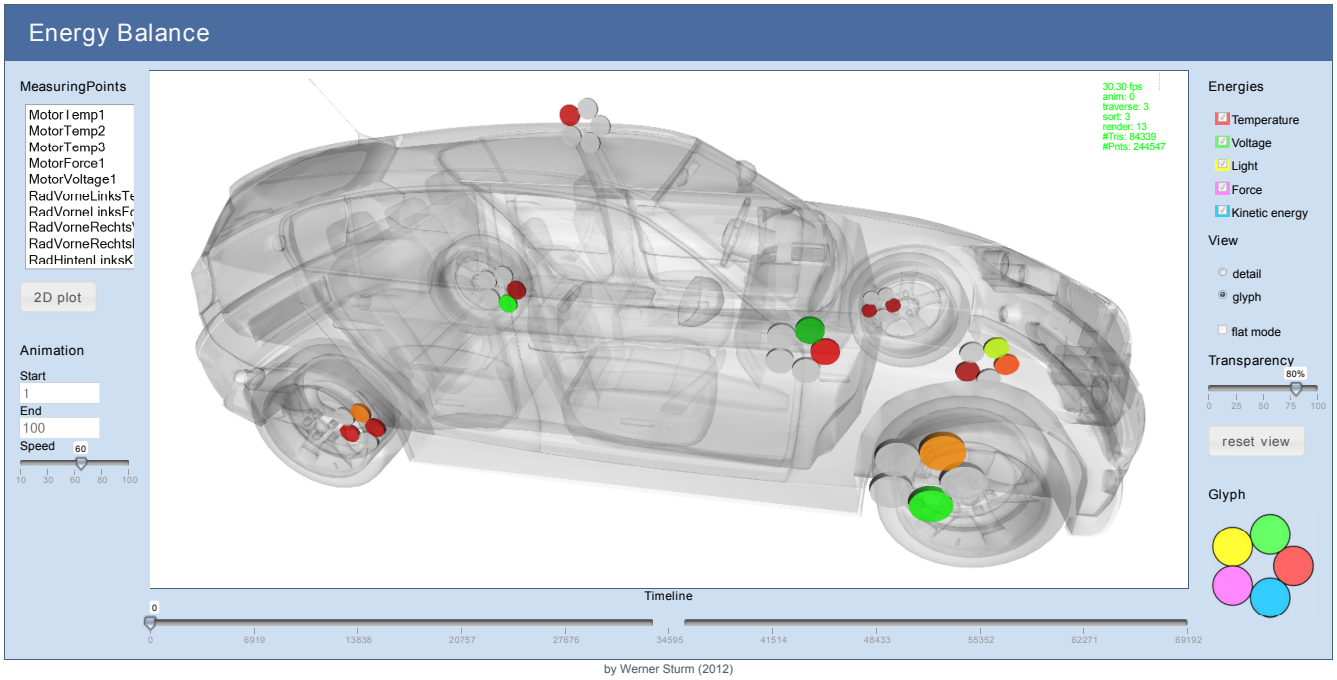


Figure 13. Screenshot of prototype displaying an overview of the data using Cumulative Glyphs.

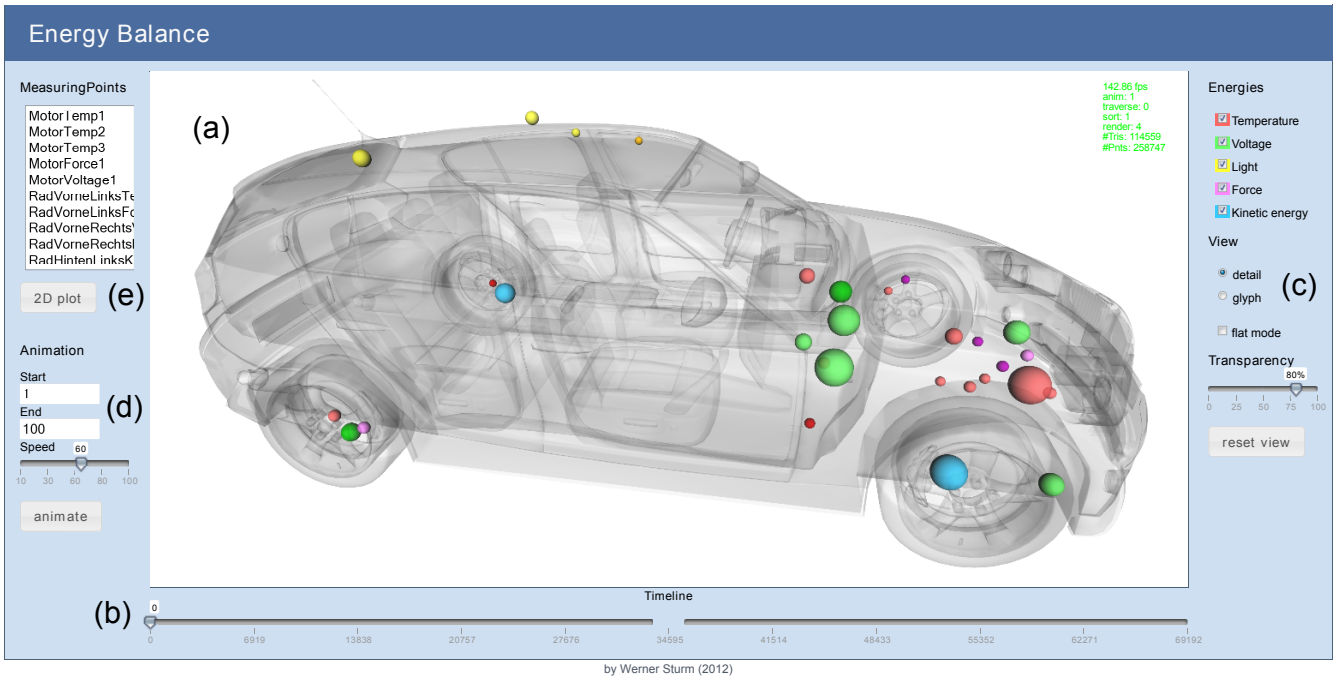


Figure 14. Screenshot of prototype displaying detailed data using Scaled Data-Driven Spheres. Real time 3D window provided by X3DOM (a). Timeline to navigate through time (b). Possibility to switch between detailed mode (SDDS) and overview mode using Cumulative Glyphs (c). Animated glyphs within specified period for three-dimensional trend analysis (d). 2D plot of values measured by selected measuring points for detailed trend analysis (e).

Moreover, D^3 can handle non-equidistant data points, which are needed to depict approximated data (as discussed in Section V-A1) without interpolating these explicitly. In addition to that, D^3 provides all required graph types (Theme River, Stacked Area Graph, Stacked Cumulative Percent Plot), which allows developers to draw graphs without the need of graph specific implementations (see Listing 5).

VI. Conclusion

This paper investigates two- and three-dimensional visualization techniques to be used to visualize consumed and emitted energies, which have been measured within a vehicle, to support automotive engineers. In addition to that, a web-based prototypal implementation of chosen visualization techniques based on Data-Driven Documents for 2D and X3DOM to perform real-time 3D client-side is introduced.

Besides the measured value and its type of energy, the multivariate data include spatial and temporal components. It revealed that Scaled Data-Driven Spheres are most suitable for a detailed 3D visualization of energy. This contributes to this research field that SDDS are also suitable for other scientific visualizations than medical visualizations. In addition to that, Cumulative Glyphs are presented to provide a better overview of the detailed data.

Both 3D-based techniques only visualize a single moment of the measurement period at once. Despite animations can represent the change over time, engineers want to analyze the trend in a more convenient way. Therefore, two-dimensional visualizations for time-oriented data are investigated in detail. Depending on the engineers requirement, Theme Rivers, Stacked Area Graphs and Stacked Cumulative Percentage Graphs support engineers to get a better understanding of the trend (see Figure 15).

A successful implementation of a prototype shows, that all techniques can be implemented as a web-enabled application without the need of installing third-party browser plugins. It revealed some issues with X3DOM like web browser incompatibilities, but most of them exist because X3DOM does not implement all specified functionalities defined by X3D standard. Despite that fact, web developers benefit from both X3DOM and Data-Driven Documents, which enable them to implement interactive visualization applications without the need to struggle with low-level graphics operations like OpenGL or drawing single lines for 2D graphs. This certainly leads to lower development costs as well.

Acknowledgment

We would like to thank the Austrian *Bundesministerium für Verkehr, Innovation und Technologie (bmvit)* for its generous support within the FEMTech project MueGen. Furthermore, we would like to thank the Institute of Automotive Engineering Graz (FTG), which provided this interesting use case and corresponding measurement data.

References

[1] W. Sturm, R. Berndt, A. Halm, T. Ullrich, E. Eggeling, and D. W. Fellner, "Energy Balance: A web-based visualization of energy for automotive engineering using X3DOM," Proceeding of the International Conference on Creative Content Technologies (CONTENT), 2013, pp. 1–10.

[2] R. Borgo, J. Kehrer, D. H. S. Chung, E. Maguire, R. S. Laramee, H. Hauser, M. Ward, and M. Chen, "Glyph-based visualization: Foundations, design guidelines, techniques and applications," M. Sbert and L. Szirmay-Kalos, Eds. Eurographics Association, pp. 39–63.

[3] G. M. Fields and R. G. Metzner, "Hybrid car with electric and heat engine," 1982, US Patent 4,351,405.

[4] J. Behr, P. Eschler, Y. Jung, and M. Zöllner, "X3DOM: a DOM-based HTML5/X3D integration model," in Proceedings of the 14th International Conference on 3D Web Technology, 2009, pp. 127–135.

[5] M. Bostock, V. Ogievetsky, and J. Heer, "D^3; data-driven documents," IEEE Transactions on Visualization and Computer Graphics, 2011, pp. 2301–2309.

[6] S. Havre, B. Hetzler, and L. Nowell, "ThemeRiver: visualizing theme changes over time," in IEEE Symposium on Information Visualization, 2000, pp. 115–123.

[7] T. Gilligan, Stacked Cumulative Percent Plots. PharmaSUG, 2009.

[8] W. Chan, "A survey on multivariate data visualization," Technical Report – Department of Computer Science and Engineering, Hong Kong University of Science and Technology, 2006.

[9] W. Aigner, S. Miksch, W. Muller, H. Schumann, and C. Tominski, "Visualizing time-oriented data–A systematic view," Computers & Graphics, no. 3, 2007, pp. 401–409.

[10] T. Ropinski, S. Oeltze, and B. Preim, "Survey of glyph-based visualization techniques for spatial multivariate medical data," Computers & Graphics, 2011, pp. 392 – 401.

[11] C. Ware, "Quantitative texton sequences for legible bivariate maps," IEEE Transactions on Visualization and Computer Graphics, 2009, pp. 1523–1530.

[12] A. A. Bokinsky, Multivariate data visualization with data-driven spots. The University of North Carolina at Chapel Hill, 2003.

[13] D. Feng, Y. Lee, L. Kwock, and R. M. Taylor, "Evaluation of glyph-based multivariate scalar volume visualization techniques," in Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization (APGV), 2009, pp. 61–68.

[14] L. Byron and M. Wattenberg, "Stacked graphs, geometry and aesthetics," IEEE Transactions on Visualization and Computer Graphics, 2008, pp. 1245–1252.

[15] Khronos, "WebGL," last accessed: June 2014, http://www.khronos.org/webgl/.

[16] K. Sons, F. Klein, D. Rubinstein, S. Byelozyorov, and P. Slusallek, "XML3D: interactive 3D graphics for the web," in Proceedings of the 15th International Conference on Web 3D Technology, 2010, pp. 175–184.

[17] jQuery Board, "JQuery," last accessed: June 2014, http://jquery.org/.

[18] T. Ropinski, M. Specht, J. Meyer-Spradow, K. Hinrichs, and B. Preim, "Surface glyphs for visualizing multimodal volume data," Proceedings of the Vision, Modeling and Visualization Workshop (VMV), 2007, pp. 3–12.

[19] M. Stone, "Choosing colors for data visualization," last accessed: June 2014, http://www.perceptualedge.com.

[20] C. G. Healey, "Choosing effective colours for data visualization," Proceedings of Visualization '96, 1996, pp. 263–270.

[21] H. Levkowitz and G. T. Herman, "Color scales for image data," IEEE Computer Graphics and Applications, 1992.

[22] B. E. Rogowitz, L. A. Treinish, and S. Bryson, "How not to lie with visualization," Computers in Physics, 1996, pp. 268–273.

[23] M. Stone, A field guide to digital color. AK Peters, Ltd., 2003.

[24] D. Borland and R. M. Taylor II, "Rainbow color map (still) considered harmful," IEEE Computer Graphics and Applications, 2007, pp. 14–17.

[25] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line for its caricature," Cartographica: The International Journal for Geographic Information and Geovisualization, 1973, pp. 112–122.

[26] K. Valkhof, "Grafico javascript charting library," last accessed: June 2014, http://grafico.kilianvalkhof.com/.

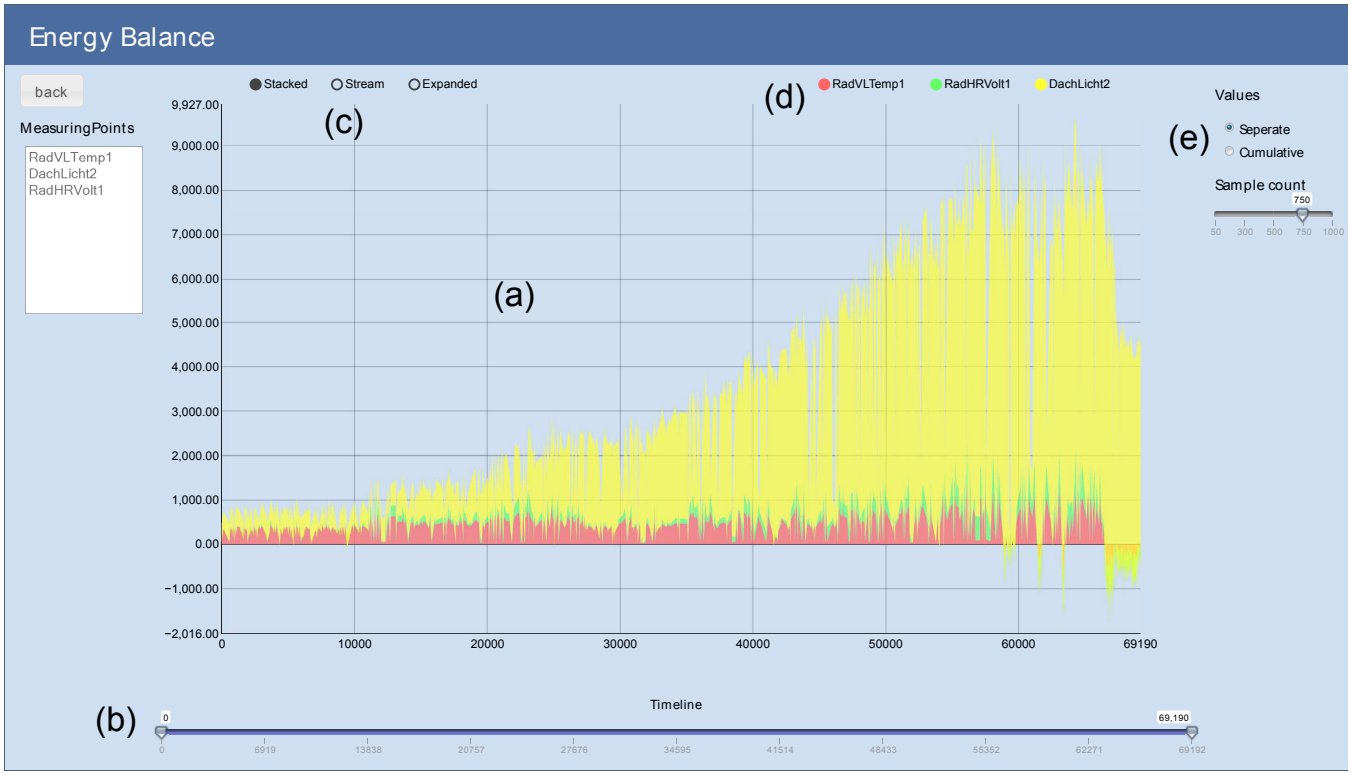


Figure 15. Screenshot of prototype showing a 2D representation of the data using Stacked Area Graphs. Graph is created using Data-Driven Documents library. Plot area for displaying the graph (a). Timeline to select a period to plot (b). Option to switch between Stacked Area Graphs (selected), Streamgraph (very similar to ThemeRiver) and Stacked Cumulative Percentage Graph (c). Possibility to select if measuring point is included in current graph (d). Switch between *seperate* and *cumulative* mode (d).

Towards Secure Mobile Computing:

Employing Power-Consumption Information to Detect Malware on Mobile Devices

Thomas Zefferer, Peter Teufl,
David Derler, Klaus Potzmader, Alexander Oprisnik, Hubert Gasparitz, and Andrea Höller

Institute for Applied Information Processing and Communications
Graz University of Technology
Inffeldgasse 16a, 8010 Graz, Austria
Email: {thomas.zefferer|peter.teufl}@iaik.tugraz.at,
{dderler|klaus.potzmader|oprisnik|hubert.gasparitz|ahoeller}@student.tugraz.at

***Abstract*—Smartphones and related mobile end-user devices represent key components of mobile computing based solutions and enable end users to conveniently access services and information virtually everywhere and any time. Due to their continuously growing importance and popularity, mobile devices have recently become a common target for malware. Unfortunately, capabilities of malware-detection applications on smartphones are limited, as integrated security features of smartphone platforms such as sandboxing or fine-grained permission models restrict capabilities of third-party applications. These restrictions prevent malware-detection applications from accessing required information for the identification of malware. This renders the implementation of reliable malware-detection solutions on smartphones difficult. To overcome this problem, we propose an alternative malware-detection method for smartphones that relies on the smartphone's measured power consumption. We show that information contained in the measured power consumption of smartphones can in principle be used to identify certain kinds of malware by means of simple threshold-based approaches. We also propose two different machine-learning techniques that allow for a classification of applications according to their power consumption in situations, where disturbing influences prevent an application of simple threshold-based approaches. The capabilities of all proposed techniques have been assessed by means of an evaluation with real-world applications running on physical smartphones. The results of this evaluation process demonstrate the applicability of power consumption based classification and malware-detection approaches in general and of the two proposed machine-learning techniques in particular.**

Keywords–Android; power consumption; application classification; malware detection; machine learning.

I. INTRODUCTION

With the growing popularity of mobile end-user devices such as smartphones or tablet computers, also malware for these devices has become an issue. The special architecture and characteristics of modern mobile end-user devices raise the demand for appropriate methods to detect such malware. Innovative approaches to detect malware on mobile end-user devices based on power-consumption measurements have been proposed by the authors in [1]. In this article, we further elaborate on the proposed techniques and on its underlying concepts.

During the past years, powerful mobile end-user devices have become part of our daily life and have significantly changed the way we access information, communicate, and interact with each other. During the past few years, smartphones and tablet computers have gradually replaced traditional end-user devices such as desktop PCs and laptops as preferred consumer devices. Considering current sales and usage statistics [2], it can be expected that mobile computing in general and smartphone-based solutions in particular will continue to play a major role in future.

The recent success of popular smartphone platforms such as Apple iOS [3] or Google Android [4] has unfortunately turned these platforms into attractive targets for attackers. This is especially problematic, as mobile end-user device typically store and process an increasing amount of security and privacy-sensitive data. In this context, malware tailored to the special characteristics of smartphone platforms has turned out to be a potential threat during the past few years. Recent reports [5] show that smartphone malware must be expected to evolve to a major issue in mobile computing in the future. By exploiting specific functionality provided by the infected smartphone platform, smartphone malware can cause financial losses, e.g., by calling premium-rate numbers or by compromising smartphone-based authentication schemes of e-banking solutions. During the past years, especially the Android platform has been frequently targeted by smartphone malware. A recent example is a malware called Eurograbber. In 2012, Eurograbber has been used to steal 47 million USD from European bank accounts by intercepting SMS-based authentication processes of e-banking portals [6]. Android seems to be especially prone to malware due to the platform's support of alternative application sources that usually lack extensive malware checks, and due to the broad functionality offered by Android's public APIs. This is advantageous for application developers and users, as it allows for mobile applications with increased functionality. At the same time, it also gives attackers the opportunity to implement more powerful malware. For instance, the Android APIs grant application developers as well as attackers full access to incoming and outgoing SMS messages, or facilitate the execution of arbitrary background tasks. While this enables the development of mobile apps that can make use of SMS functionality, it also facilitates the development of malware that intercepts or spies on incoming

messages.

The factual vulnerability of the Android platform against malware raises the need for reliable methods to distinguish benign apps from malicious ones and to detect unwanted behavior on smartphones. On classical end-user devices such as desktop computers or laptops, this functionality is typically implemented by anti-virus software, which is able to detect malicious software at runtime. Unfortunately, the deployment of anti-virus software on smartphone platforms in general, and on Android in particular is difficult. This is mainly due to the fact that Android (as well as other smartphone platforms) implements several security features on operating-system level, which limit access rights and capabilities of third-party applications. For instance, all smartphone applications are executed in a so-called sandbox and are, thus, unable to access resources of other applications being installed and executed on the same device. This way, application-specific data remains protected from unauthorized access by other applications. While implemented security features definitely improve the system's basic security, they also render the implementation of supplementary security software difficult. For instance, the implemented sandbox feature prevents anti-virus software on Android smartphones from collecting information that is required to reliably detect smartphone malware at runtime.

Integrated security features that limit the capabilities of classical malware-detection methods can theoretically be bypassed by rooting the smartphone's operating system. Rooting has become common practice in the power-user community, as it gives users more control over the device and allows for additional functionality. However, rooting is not really an option to increase the capabilities of anti-virus software, as it significantly decreases the smartphone's overall security and enables additional attack vectors. Furthermore, non-rooted device still represent the majority of all mobile end-user devices. For these reasons, we focus on the class of non-rooted devices only.

The reliable detection of malware on non-rooted smartphones is still an unsolved problem that definitely needs to be addressed to assure the security of future mobile computing. To overcome this problem, we propose a new technique to detect malware on mobile end-user devices. The proposed technique compensates the lack of required information about running applications by making use of side-channel information being available on non-rooted Android smartphones. This way, this work answers two basic research questions. First, this work evaluates if and to what extend power-consumption information available on smartphones can be used to classify running mobile applications and to identify malware. Second, this work investigates capabilities of different machine-learning techniques to analyze available power-consumption information.

The results presented in this article extent first results that have been presented in 2013 [1]. The obtained results show that alternative approaches to identify malware on mobile end-user devices can be successful. Furthermore, obtained results indicate that the application of machine-learning techniques can improve the classification of applications according to their power consumption. Even though the described work is basically a first proof of concept, the obtained results are promising and open new possibilities for malware detection on mobile end-user devices. This way, this work contributes to the security of future mobile-computing applications.

The remainder of this paper is structured as follows. In Section II, existing malware-detection approaches for smartphones are briefly surveyed and limitations of these approaches are identified. Subsequently, Section III introduces the tool PowerTutor [7] and explains our approach to measure the power consumption of applications on smartphones. In Section IV, we show that measured power-consumption traces can indeed be used to enhance the security of critical applications on smartphones by means of a simple detection mechanism for SMS-based malware. In Section V, we focus on more sophisticated methods to analyze collected power-consumption information and propose two methods to analyze collected power-consumption measurements based on approved machine-learning techniques. We evaluate the capabilities of the proposed methods to classify applications and to distinguish benign applications from malicious ones in Section VI. Finally, conclusions are drawn in Section VII.

II. Related Work

During the past years, several approaches to detect and analyze malware on mobile platforms have been introduced. Basically, existing approaches can be classified into static and dynamic analysis methods. Static analysis refers to the inspection of an application's source code or binary package without running it, whereas dynamic analysis involves running the application to capture additional information.

Dynamic analysis includes techniques such as Information Flow Analysis, where private data is labeled and prevented from leaving the device. TaintDroid [8] is an Android kernel extension that follows this approach and allows for dynamic taint tracking. Dynamic approaches, which apply machine-learning techniques to distinguish benign applications from malicious ones, include [9] by Shabtai et al. and [10] by Burguera et al. Both references include extensive listings of related Android-based malware-detection systems. Most of these approaches run candidate applications in a sandbox to derive measurements, such as system-call intervals and networking usage. Our technique presented in this paper follows a similar approach. Similar to related work, we make use of side-channel information to classify smartphone applications. However, in contrast to other related work, we rely on power-consumption measurements for this purpose.

A comprehensive overview of dynamic malware-analysis techniques is also provided by Egele et al. in [11]. Many of these methods are highly advanced in detecting and analyzing malware. However, these methods usually require complex external analysis frameworks and can hardly be deployed on non-rooted end-user devices, due to their requirement to deeply integrate into the smartphone's operating system. Thus, these methods are usually less useful for detecting malware on typical end-user devices at runtime.

The technique presented in this paper addresses this problem and facilitates dynamic malware detection directly on non-rooted Android phones by analyzing the devices' power consumption. A related approach to analyze the power-consumption in order to detect malware has been followed by Jacoby and Davis [12], who have proposed an intrusion

detection system that correlates various attack scenarios to typical power consumptions. Additional work has been published by Buennemeyer et al. [13] [14], who propose systems, which use power profiles of phones to detect malware targeting battery drainage. Our technique follows a similar approach but extracts more detailed information from the collected power-consumption measurements in order to classify applications and to detect malware.

Employing the power consumption to reveal additional information on an IT system is actually not a new idea. For instance, approaches to extract secret information stored on smart cards by means of measuring and analyzing their power consumption have already been proposed in 1999 by Kocher et al. [23]. Based on this work, various techniques to reveal secret data and information from IT systems have been proposed during the past few decades. Most of these techniques however require an elaborate measurement equipment and the application of complex analysis methods. This renders a real-time application of these approaches on current mobile end-user devices difficult. The techniques proposed in this paper follow a slightly different approach and rely on methods that basically allow for real-time application.

For the proposed techniques, the collection of accurate power-consumption measurements on smartphones is a key aspect and mandatory enabler of our technique. We discuss details of this aspect in the next section.

III. Measuring the Power Consumption of Smartphones

Measurements of a smartphone's power consumption build the basis of the proposed classification and malware detection techniques. To collect the required power-consumption measurements, we rely on the tool PowerTutor by Zhang et al. [7]. Another tool that would allow for the acquirement of this kind of information is Trepn [15]. In contrast to PowerTutor, Trepn uses hardware sensors and thus promises more exact measurements. However, Trepn is limited to the Snapdragon mobile development platform [16] and can hence not be applied on typical Android based end-user devices. Heading for a solution that is applicable on real end-user devices, PowerTutor has therefore been our tool of choice.

The tool PowerTutor is basically a smartphone application that measures the power consumption of all applications running on the same smartphone. For each application, the power consumption of six smartphone components is measured. In particular, the power consumption of the components *CPU*, *Audio*, *Display*, *Wi-Fi*, *3G* and *GPS* is measured separately. Figure 1 shows the mean per-component power consumption of six different applications. All information shown in Figure 1 has been directly derived from measurements created by the tool PowerTutor. The graphically prepared measurements shown in Figure 1 clearly indicate that there are significant differences between power-consumption measurements of different applications. Furthermore, Figure 1 shows that there are also differences in the power consumption of different components. Unsurprising, the smartphone display and the mobile network (3G) are responsible for most of the measured power consumption - at least for the six measured applications[1].

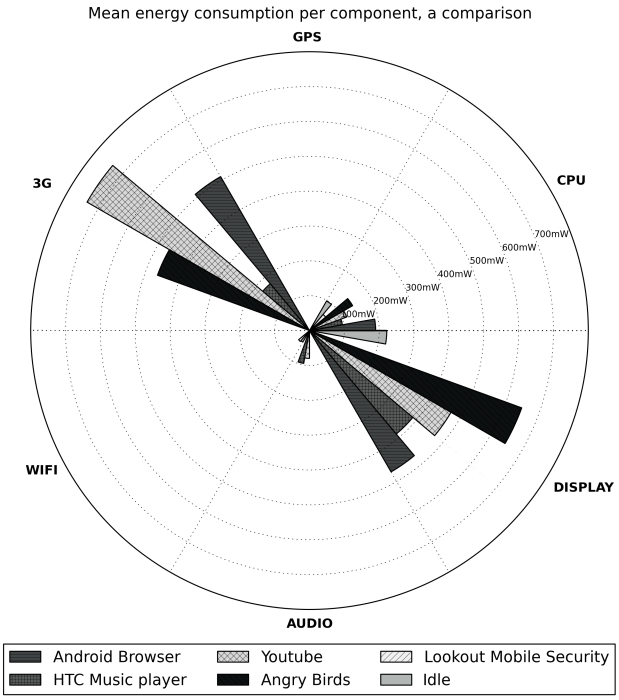


Figure 1: Comparison of mean power usage per component using six different applications

Figure 1 shows that power measurements provided by the tool PowerTutor indeed contain valuable information that might allow for a classification of running applications and for an on-the-fly identification of malicious apps at runtime. To improve efficiency and to appropriately cope with limited processing power on mobile end-user devices, we have decided to focus on measurements of one component only in a first step. Analysis of power-consumption measurements of several applications have shown that the smartphone CPU is actually the best suited component for profiling running applications by means of its power consumption. This is also illustrated by Figure 2. This figure shows the stacked power consumptions of the six measured components while running the app *HTC Music Player*. This figure shows that the CPU is basically the only component that shows a significant change in power consumption over time. Similar results have also been obtained for other applications, which justifies our decision to focus on the power consumption of the CPU only. At this point, it has to be noted that ignoring all other components of course reduces the amount of available information. However, considering all available power-consumption information from all components significantly increases complexity and is hence considered as future work.

Figure 3 shows the measured power consumption of the CPU component caused by the smartphone game Angry Birds, the Android Browser, the Idle process, and the security application Lookout Mobile Security. These measurements show

[1]GPS is also know to consume significant amounts of power. However, none of the six measured applications used GPS functionality during the measurements.

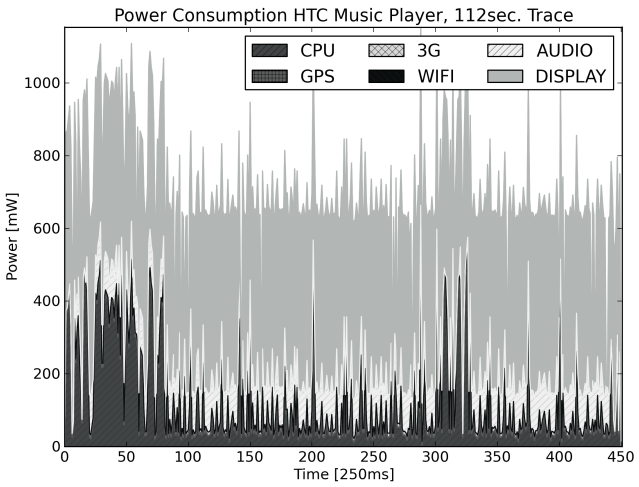


Figure 2: Power usage of HTC Music Player stacked per component.

that there are obvious differences in the measured power consumption of different applications running on the same device. Obviously, the application Angry Birds and the Web browser consume significantly more power than the Idle process or the application Lookout Mobile Security. The first question that arises from these observations is whether and to what extend these differences in the power consumption can be used to identify applications and suspicious behavior. We show in the next section that this question can be positively answered and that differences in measured power consumptions can in some cases be directly used to identify applications with suspicious behavior on smartphones at runtime.

IV. Detection of SMS-Controlled Spyware

SMS-controlled spyware has evolved to a serious threat on the Android smartphone platform. This kind of malware spies on private user data (e.g., position information, data received and sent via SMS, etc.) and is able to receive hidden control commands via SMS. These messages contain control commands and are sent to the victim's smartphone unnoticed by the legitimate user. Apart from spyware, capturing and processing SMS messages is also handled by malware (e.g., Eurograbber), which attacks two-factor authentication systems that are usually employed by online-banking systems. Spyware and malware related to SMS functionality are typically implemented for the Android platform, due to the availability of public APIs for accessing low-level SMS functionality. This is in contrast to iOS and Windows Phone, where this functionality is only handled by the operating system and cannot be used via public APIs.

In general, spyware and malware, which access SMS-related APIs can be assigned to two different categories according to the implemented functionality.

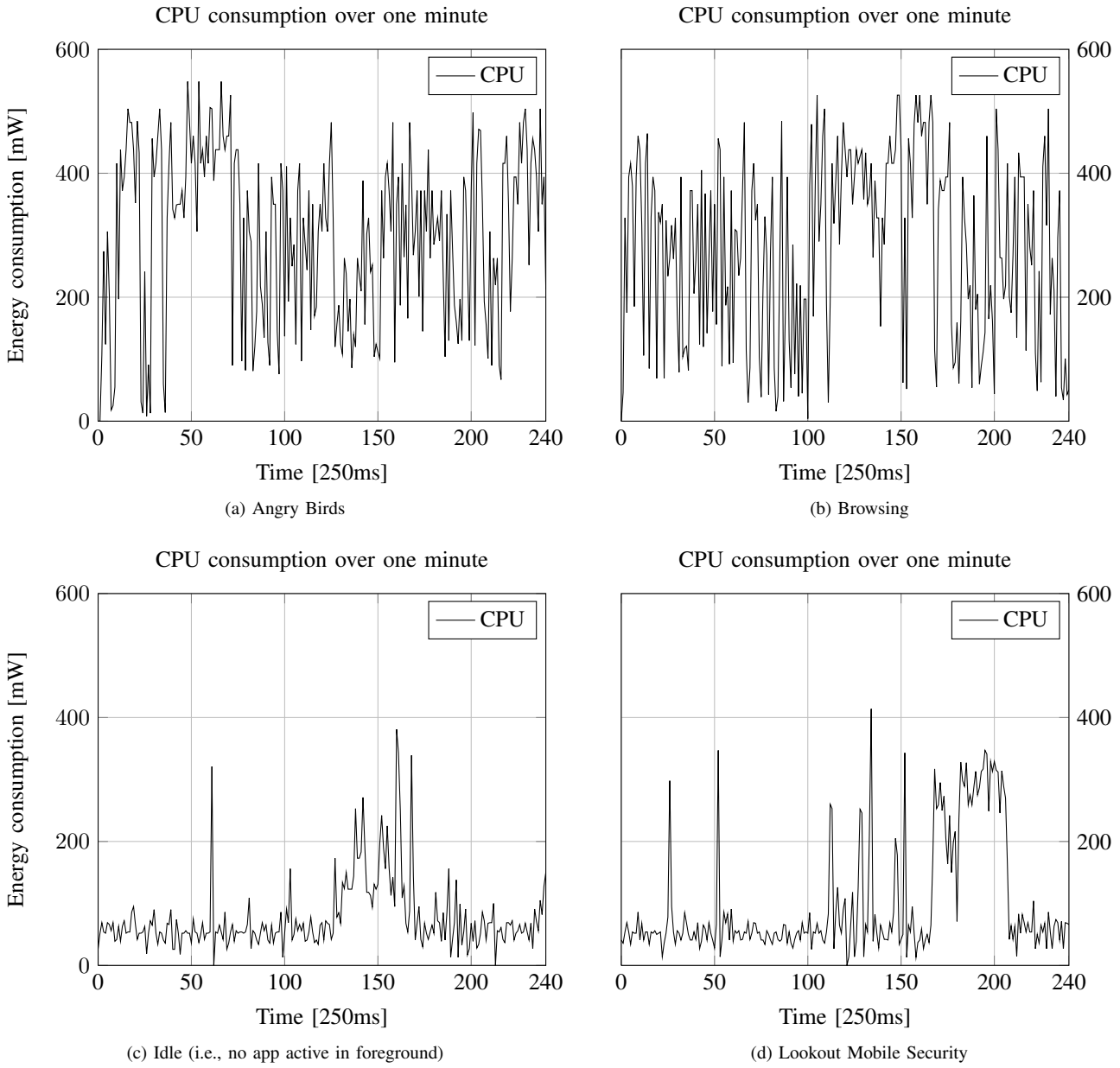
- **SMS Sniffers:** SMS sniffers, which are mostly relevant for spyware, only capture the information in the received SMS messages, and process and optionally forward this information.
- **SMS Catchers:** SMS catchers also suppress the SMS forwarding mechanism of the operating system, which would deliver the message to the default SMS client. Therefore, the user will not receive a notification on the newly arrived messages. This functionality is often required by malware that attacks online banking systems, because the received TAN must not be shown to the user, who would otherwise be alarmed by receiving a TAN without executing an actual transaction. The problem of SMS catchers has recently been addressed by Android 4.4., which requires definition of a default SMS application that always receives incoming messages without the possibility to suppress the notification of the user. However, the market share of Android 4.4. is still very limited and the principle problem of reading SMS messages remains.

To assess the general potential of power consumption based malware-detection approaches, we show how to identify suspicious SMS processing activities by analyzing the power consumption of an Android smartphone. For this purpose, we have developed the following two smartphone applications for the Android platform:

- **Malware Simulator:** This application simulates SMS-controlled spyware and is able to intercept incoming SMS messages as well as to silently forward copies of received SMS messages to arbitrary recipients. Furthermore, this application is able to determine the smartphone's current location either by reading out the last known position from the Android system, or by determining the current position using available position sensors. The determined position information can be forwarded to an attacker by SMS. This way, the developed malware simulator implements all typical features of SMS-controlled spyware, which receives commands via SMS, spies on the user's current location, forwards determined location information, and silently forwards copies of SMS messages to a remote attacker.
- **Malware Detector:** This application detects anomalies of local SMS processing activities by measuring and analyzing power-consumption information provided by PowerTutor. Thus, the basic goal of this application is to successfully identify the implemented **malware simulator** as spyware.

To evaluate the general capabilities of power consumption based malware detection approaches, we carried out the following steps: First, the malware detector has measured the power consumption for three seconds each time an SMS message has been received by and processed on the device. Second, from the obtained measurements, an appropriate model has been extracted. According to this model, all measured power traces have been classified into four categories depending on the SMS processing operation that has been taking place during the measurements. Finally, the extracted model has been integrated into the developed malware detector. With the help of this model, the malware detector has then been used to identify suspicious activities during SMS processing operations in real time.

Following this approach, both the malware simulator and



(a) Angry Birds

(b) Browsing

(c) Idle (i.e., no app active in foreground)

(d) Lookout Mobile Security

Figure 3: One minute plots of CPU energy consumption

the malware detector have been deployed on a Samsung Galaxy S2 smartphone running the Android 2.3 operating system. The device's power consumption has been measured during execution of the following SMS processing steps carried out by the malware simulator:

- **Step A – Normal SMS receive**: An incoming SMS message is received and forwarded to Android's default SMS application without any further action.
- **Step B – Pos. Command 1**: An incoming command SMS message is intercepted and an SMS message is returned that includes the last known location.
- **Step C – Pos. Command 2**: An incoming command SMS message is intercepted and an SMS message is returned that includes the currently determined location.
- **Step D – Forward SMS**: An incoming SMS is forwarded to Android's default SMS application and additionally forwarded to a given number.

The power consumption of each processing step has been measured for ten times. Figure 4 shows the average power consumption of the four different SMS processing steps. Obviously, there are significant differences in the power consumption depending on the executed processing step. If the SMS is forwarded to Android's default SMS application for further processing (**Step A** and **Step D**), more energy is consumed compared to processing steps, in which an incoming SMS message is intercepted and another SMS message is sent unnoticed by the user (**Step B** and **Step C**). Due to the

significant differences in the measured power consumptions, an appropriate model for the detection of suspicious activities can be extracted by simply considering the average value of the power measurements. In the present case, an average power consumption of 150 has been chosen to define the lower bound of an SMS that is processed by Android's SMS application. If the average value is beneath this bound, it is likely that an incoming SMS message has been intercepted and hidden from the user.

The reliability of this rather simple model has been evaluated by sending 10 normal SMS messages that have simply been forwarded to Android's SMS application. Additionally, we have sent ten SMS messages, that have been intercepted by our malware simulator and discarded afterwards. By measuring the power consumption during execution of the SMS processing steps and by applying our simple model to the obtained measurements, the implemented malware detector was able to successfully distinguish between discarded messages and messages forwarded to Android's SMS application. This way, we have shown that it is basically possible to detect in real time whether an SMS message is received normally or whether it is intercepted by a third party application. This validates the postulated assumption that information contained in power-consumption measurements can indeed be used to identify malicious applications on smartphones and hence positively answers the first research question of this work.

V. Enhanced Classification Techniques

The successful realization of a malware detector that is able to identify suspicious SMS processing activities at runtime confirms the capabilities of application-classification and malware-detection techniques based on power-consumption information. However, even though there are obvious differences in the power consumption of these two applications (Figure 3), an immediate identification and classification of applications based on such measurements is often not possible. This is due to the fact that the measured power consumption is not only influenced by the application itself, but also by other effects, such as varying user inputs, the processed data, different screen orientations, the deployment of hardware acceleration techniques, or 3G or WiFi signal reception. This is illustrated in Figure 5, which shows two different measurements of one and the same application. Although stemming from the same application, the two measured power-consumption traces are quite different.

Disturbing influences render the determination of a simple and unique power-consumption signature for a given application or smartphone state impossible. Unambiguous results, such as the ones obtained for the implemented malware detector for identification of SMS-controlled spyware, are usually hard to achieve in practice. To overcome this problem, we propose two analysis techniques that rely on approved machine-learning approaches. The proposed techniques can be used to classify smartphone applications according to their power consumption, even if there are only minor differences in collected power measurements due to disturbing influences.

During the past years, different machine-learning techniques for the classification of data have been proposed. For the given scenario, i.e., the classification of smartphone applications based on their power consumption, two techniques have been chosen and adapted to the given requirements. Both techniques consist of a *learning phase* and a *classification phase*. During the learning phase, well-known input data is used to train a model. In the subsequent classification phase, the trained model is used to classify unknown input data. The two techniques are discussed in more detail in the following subsections.

A. Power-Consumption Histograms

This technique is rather simple and counts how often a specific application is on a certain power-consumption level. In order to model this, we have computed power histograms by dividing the interval between 0% power consumption and 100% power consumption into 15 disjoint and equal-sized intervals. A histogram is then created by simply assigning each data point to exactly one interval and counting the data points in each interval. In order to cope with differences in the absolute power consumption, the values have been normalized appropriately. During the learning phase, the average histograms have been created by measuring the power consumption of well-know applications. Figure 6 shows some examples of average histograms for different applications that have been obtained during the training phase.

In the classification phase, the histograms of applications to be classified are compared with the trained average histograms by applying distance-measures such as cosine similarity. To assess the capabilities of this approach, this technique has been evaluated in a real-world scenario. Results of this evaluation process are presented and discussed in Section VI.

B. MFC Coefficients and Gaussian Mixture Models

This technique makes use of *Mel Frequency Cepstral Coefficients (MFCC)* to classify smartphone applications based on their power consumption. This technique has originally been introduced for speaker-recognition systems [17][18] and is also frequently used for music similarity finders [19][20]. In such systems, MFC coefficients and their distribution are extracted from recorded voice or music using complex transformations as implemented by the *melcepst* function [21]. The distributions of the extracted MFCC are then used to create a *Gaussian Mixture Model (GMM)* for each MFCC. The resulting GMM define a unique representation of the recorded voice or music. Later recordings of voice or music can be compared to existing representations in order to implement voice-recognition and music-similarity finders.

Our intention behind using a speaker recognition approach was to map the problem of matching voice recordings to a person to the problem of matching power measurements to an application. Spoken voice recordings vary in pitch and frequency and are very unlikely to be equal between two recordings. This, naively speaking, resembles the problem we face with power-consumption measurements.

Our implementations bases on an existing speaker-recognition implementation by Anil Alexander [22]. This implementation relies on GMM and MFCC and can be customized with a number of parameters including the number of Gaussians and the number of MFCC to use. Experiments have shown that for our purposes best results can be achieved

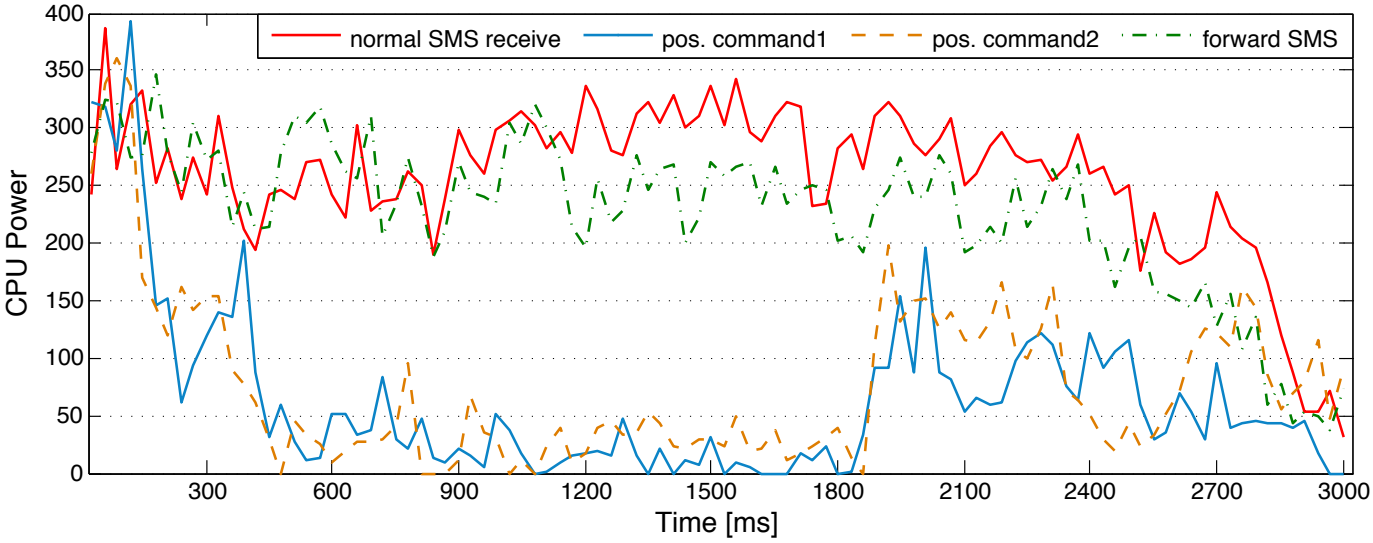
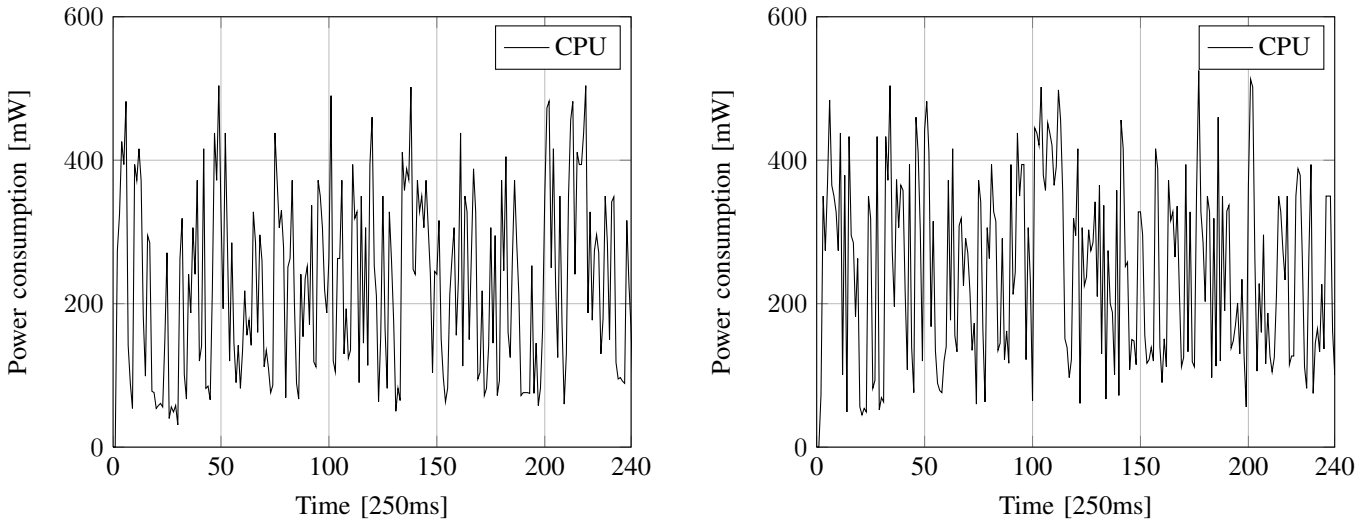


Figure 4: Average CPU power profile after receiving an SMS



(a) Angry Birds - Run 1

(b) Angry Birds - Run 2

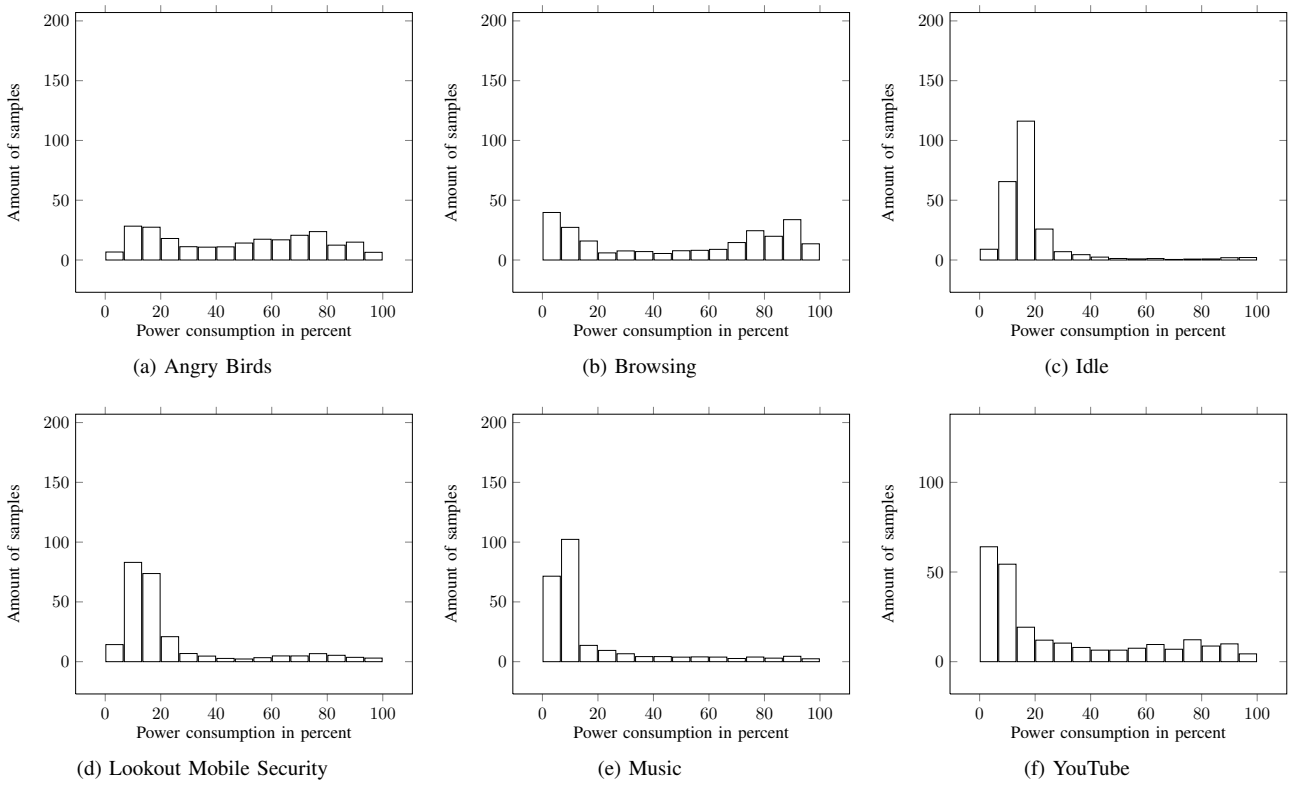
Figure 5: One minute plots of CPU energy consumption

with three Gaussians and twelve MFCC. Hence, during the learning phase the distributions of twelve MFCC are computed from power-consumption measurements for each class of application. The computed distributions of the twelve MFCC are then approximated using a GMM with three Gaussians. The resulting GMM finally represents the result of the learning phase. Figure 7 illustrates the distribution of twelve different MFCC and the resulting GMM.

During the classification phase, MFCC are derived from power-consumption measurements of the application to be classified. For each derived MFCC, the best matching GMM is selected out of all GMM that have been obtained during the learning phase. By combining the classification results of all twelve MFCC, the best matching application class is finally determined.

VI. Evaluation

We have evaluated the reliability and efficiency of the proposed feature extraction techniques by testing prototype implementations of the two techniques in a real-world scenario. Required power-consumption measurements have been acquired using the tool PowerTutor. For convenience reasons, the classification itself has been performed off the mobile device, as the learning phase (especially for the speaker recognition based approach) is rather slow. This subsection describes the model that has been used to classify applications, discusses details of the dataset creation, and presents results that have been obtained by applying the two classification techniques introduced in Section V.



(a) Angry Birds (b) Browsing (c) Idle

(d) Lookout Mobile Security (e) Music (f) YouTube

Figure 6: Average histograms of different applications

A. Classification Model

Applications with the same or almost the same purpose are expected to cause similar power consumptions. Therefore, we have roughly grouped applications into distinct sets according to their purpose. The resulting list of groups is no comprehensive classification scheme of all available applications. It is merely a logical grouping of the power-consumption measurements we gathered in this experiment and does raise no claim to completeness. Based on the gathered measurements, the following six groups of applications have been defined: *Games*, *Internet*, *Idle*, *Malware*, *Music*, and *Multimedia*. Note that malware and security software have been assigned to the same group. Both malware and security software usually remain idle in the background until being activated by a certain event (e.g., reception of a command message via SMS). This comparable behavior leads to a comparable power consumption too and justifies a common classification of these both types of application. Of course, also other malware with different behavior and hence a different power-consumption profile exists. However, for a first proof of concept only malware with the above-described behavior has been considered. Consideration of other types of malware is regarded as future work.

B. Dataset Creation

PowerTutor provides specific measurements for each running application. However, in practice these application specific measurements have turned out to be not as reliable and accurate as desired. Therefore, we have refrained from using application specific measurements and have relied on system-wide power-consumption measurements provided by PowerTutor instead.

We have further limited subsequent analysis steps to the measured power consumption of the smartphone's CPU. Although PowerTutor also provides measurements for other smartphone components such as the display or the GPS receiver, measurements of these components have been omitted in order to reduce computation costs when learning and due to the fact that these components often lack activity.

To evaluate the proposed classification techniques, we finally created 96 system-wide power-consumption measurements (CPU) using a customized instance of PowerTutor. To facilitate a subsequent analysis, we have adapted PowerTutor such that beside the measurement values themselves also the device model, the capture date, and the sample rate have been stored. The 96 captured measurements (sixteen measurements per application group) have been limited to the length of about one minute, with a total of 247 data points per measurement. We have cut off the trace length after about one minute, as this is a realistic time-frame for real-world scenarios. Analysis of longer measurements is considered as future work. Similarly, increasing the number of analyzed applications is also considered as future work. For a first proof of concept, the used number of applications and the chosen time interval is however sufficient. In total, six devices have been used to collect the measurements (three Samsung Galaxy S2 smartphones and three HTC Desire devices). To reduce noise, only the application to be measured and PowerTutor have been active during the measurements.

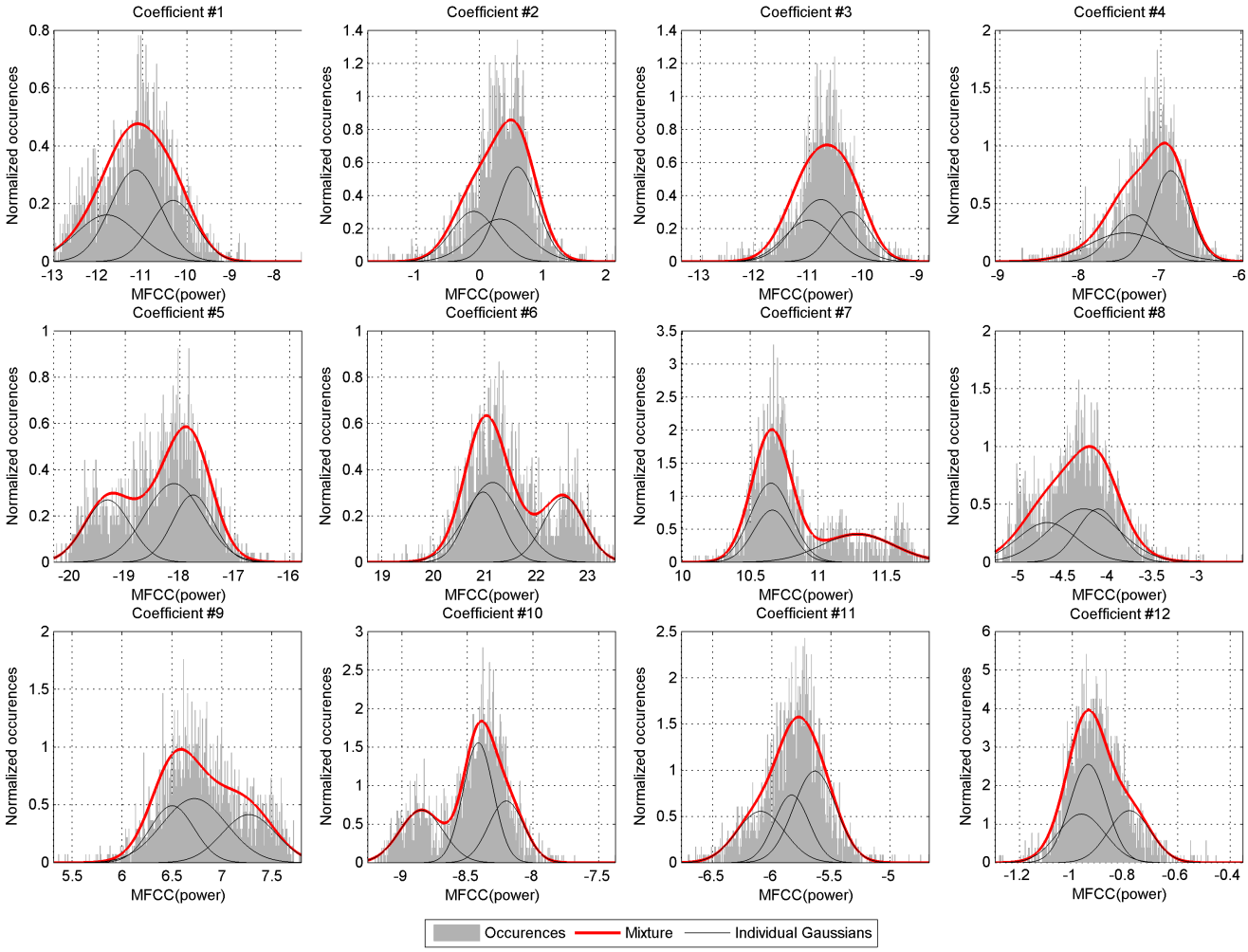


Figure 7: Gaussian Mixture Models of twelve MFCC derived from power-consumption measurements

C. Results

The 96 captured measurements have been used to evaluate the efficiency and reliability of the proposed classification techniques. As quality indicators, the positive predictive value (PPV, also referred to as precision), the true positive rate (TPR, also referred to as recall or sensitivity), the true negative rate (TNR, also referred to as specifity), the accuracy, and the area under the receiver operating characteristic (AUC) have been used. According to its definition, PPV refers to the correct positive classification in relation to all positive classifications. Accordingly, TPR refers to true positives given all real positives. TNR denotes true negatives (TN) given all negatives. Accuracy is the relation between correctly classified samples given all samples. The receiver operating characteristic is a graphical representation of the trade-off between TPR and FPR (1-TNR). AUC (also sometimes denoted as AUROC) refers to the area below this resulting curve.

In order to appropriately divide the available measurements in training and test data, we have folded the available dataset using 10-fold cross validation. To enhance the robustness of the obtained results, average values over 100 runs are presented.

TABLE I: HISTOGRAM BASED APPROACH: CONFUSION MATRIX FOR CATEGORIES GAMES (G), INTERNET (IN), IDLE (ID), MALWARE (MW), MUSIC (MU), AND MULTIMEDIA (MM)

	G	IN	ID	MW	MU	MM
G	13.98	1	0	1.02	0	0
IN	1	13.14	0	0	0	1.86
ID	0	0	12	4	0	0
MW	0	0	3.97	10.07	1.96	0
MU	0	0	0	2	9.24	4.76
MM	0.27	0.75	0	0	0	14.98

For our performance evaluation, a confusion matrix for the six predefined application categories has been created, which can be interpreted in the following way: Values in the diagonal of the matrix have been classified correctly (true positives), values within a row not in the diagonal represent false negatives and values within a column not in the diagonal represent false positives. Other values are considered true negatives.

Obtained results of the histogram based approach are

TABLE II: CLASSIFICATION RESULTS (HISTOGRAM BASED APPROACH)

Category	PPV	TPR	TNR	Accuracy	AUC
Games	0.87	0.92	0.98	0.97	0.95
Internet	0.82	0.88	0.98	0.95	0.93
Idle	0.75	0.75	0.95	0.92	0.85
Malware	0.63	0.59	0.91	0.87	0.75
Music	0.58	0.83	0.98	0.91	0.90
Multimedia	0.94	0.69	0.92	0.92	0.80

TABLE III: MFCC AND GMM BASED APPROACH: CONFUSION MATRIX FOR CATEGORIES GAMES (G), INTERNET (IN), IDLE (ID), MALWARE (MW), MUSIC (MU), AND MULTIMEDIA (MM)

	G	IN	ID	MW	MU	MM
G	10.40	3.46	0.01	0	0.55	1.58
IN	2.07	12.67	0	0	0.26	1
ID	0.39	0	11.65	3.6	0.18	0.18
MW	0.07	0.01	2.56	13.15	0	0.21
MU	0.22	0.81	0	0.97	9.39	4.61
MM	0.96	2.97	0.01	0.03	1.20	10.83

shown in Table I and Table II. In case of the MFCC based approach, best results have been achieved with 3 Gaussians and twelve MFCC. The performance evaluation results of the MFCC based approach are outlined in Table III and Table IV.

D. Discussion

From these results, various findings can be derived. Mobile security applications and malware running in the background can generally be distinguished from application being active at the moment (with the exception of system services). Games, Internet, music and multimedia applications are distinguishable as well. Music and multimedia applications are more difficult to distinguish correctly, due to their similar power-consumption profile. However, given their related purposes this is plausible. Streaming a YouTube video with sound is not too different from listening to music while reading related information displayed by the music player.

The obtained results have also revealed that the MFCC based approach works better for the distinction between the categories Idle and Malware. Therefore, this approach seems to be more suitable for malware-detection purposes. On the other hand, the histogram approach constitutes a fast classification method, suitable for mobile devices with limited computational power.

It has to be noted that the obtained results basically represent a first proof of concept only. The number of measured applications and also the time period, in which the power consumption of applications has been measured, has been intentionally kept rather low for the sake of simplicity and both, an analysis regarding a larger dataset and an analysis regarding a larger number of applications is left open for future work.

Although the number of applications and also the length of measurements has been fixed to a relatively small value, obtained results are still useful due to using 10-fold cross validation, and, thus, answer the second research question of this work. Concretely, obtained results show that machine-learning techniques are suitable to analyze power-consumption measurements of smartphone applications for classification and malware-detection purposes.

TABLE IV: CLASSIFICATION RESULTS (GMM BASED APPROACH)

Category	PPV	TPR	TNR	Accuracy	AUC
Games	0.65	0.74	0.95	0.90	0.85
Internet	0.79	0.64	0.91	0.89	0.78
Idle	0.73	0.82	0.97	0.93	0.90
Malware	0.82	0.75	0.94	0.92	0.85
Music	0.59	0.82	0.97	0.91	0.90
Multimedia	0.68	0.59	0.91	0.87	0.75

VII. CONCLUSION

Smartphones and related mobile end-user devices are frequently used to store and process security and privacy-critical data. Malware on smartphones is a growing threat for these data and hence a major challenge for future mobile computing solutions. To overcome this challenge, new and innovative methods to detect malware on smartphones and related mobile end-user devices are needed. In this paper, we have tested the hypothesis that the power consumption of smartphones correlates with the kind of applications being executed on the smartphone and that this correlation allows for a classification of applications and a detection of malicious software. To test this hypothesis, we have proposed a simple threshold-based method and two machine-learning techniques that can be used to classify unknown applications according to their power consumption. We have further assessed the validity of the general hypothesis and the capabilities of the proposed machine-learning techniques by means of a concrete prototype implementation and a succeeding evaluation in a real-world scenario. The conducted assessment has corroborated the constructed hypothesis and has shown the capabilities of the proposed techniques to correctly classify smartphone applications according to their power consumptions.

Although first results are promising, this work mainly represents a proof of concept and a solid basis for future work. In a next step, we plan to port the entire classification onto a smartphone in order to render external classification frameworks unnecessary. Power measurements can already be collected directly on the smartphone using tools such as PowerTutor. The development of a purely smartphone based application classification and malware detection solution that relies on the techniques presented in this paper is hence mainly a matter of computing resources available on smartphones. Since information on the smartphone's power consumption is publicly available on Android smartphones, our solution does not require root access to the operating system and is hence applicable on virtually all end-user devices. We are also planning to refine the proposed techniques and to enhance the current prototype in order to achieve even more accurate results and to be able to classify multiple applications running simultaneously on a smartphone.

REFERENCES

[1] T. Zefferer, P. Teufl, D. Derler, K. Potzmader, A. Oprisnik, H. Gasparitz and A. Hoeller, "Power consumption-based application classification

and malware detection on Android using machine-learning techniques," *Future Computing, 2013. Proceedings from the fifth international conference on future computational technologies and applications*, pp. 26–31, May-June 2013.

[2] Go-Gulf, "Smartphone Users Around the World - Statistics and Facts," http://www.go-gulf.com/blog/smartphone/, 2013.

[3] Apple, "Apple iOS 6," http://www.apple.com/ios/, 2013.

[4] Google, "Android," http://www.android.com/, 2013.

[5] Lookout Mobile Security, "2011 Mobile Threat Report," https://www.mylookout.com/mobile-threat-report, 2011.

[6] InformationWeekSecurity, "Zeus Botnet Eurograbber Steals $47 Million," http://www.informationweek.com/security/attacks/zeus-botnet-eurograbber-steals-47-millio/240143837, 2012.

[7] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 105–114.

[8] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[9] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, Feb. 2012.

[10] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '11. New York, NY, USA: ACM, 2011, pp. 15–26.

[11] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, pp. 6:1–6:42, Mar. 2008.

[12] G. A. Jacoby, R. Marchany, and N. J. Davis, "Battery-based Intrusion Detection: A First Line of Defense," *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pp. 272–279, Jun. 2005.

[13] T. K. Buennemeyer, T. M. Nelson, L. M. Clagett, J. P. Dunning, R. C. Marchany, and J. G. Tront, "Mobile device profiling and intrusion detection using smart batteries," in *Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, 2008, p. 296.

[14] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proceedings of the 6th international conference on Mobile systems, applications, and services*, ser. MobiSys '08. New York, NY, USA: ACM, 2008, pp. 239–252.

[15] Ben-Zur, Liat, "Developer Tool Spotlight - Using Trepn Profiler for Power-Efficient Apps," https://developer.qualcomm.com/blog/developer-tool-spotlight-using-trepn-profiler-power-efficient-apps, October 2011.

[16] Bsquare, "Snapdragon Based Products and Services," http://www.bsquare.com/products/snapdragon-based-products-and-services, 2013.

[17] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker Verification Using Adapted Gaussian Mixture Models," *Digital Signal Processing*, vol. 10, no. 1-3, pp. 19–41, 2000.

[18] Kumar, G Suvarna and Raju, K.A. Prasad and Rao, Mohan and Satheesh, P, "Speaker Recognition using GMM," *International Journal of Engineering Science*, vol. 2, no. 6, pp. 2428–2436, 2010.

[19] B. Logan and A. Salomon, "A music similarity function based on signal analysis," in *ICME*, 2001.

[20] K. C. West and P. Lamere, "A model-based approach to constructing music similarity functions," *EURASIP J. Adv. Sig. Proc.*, vol. 2007, 2007.

[21] M. Brookes, "VOICEBOX: Speech Processing Toolbox for MATLAB," Web page, 2005.

[22] A. Alexander, "Automatic Speaker Recognition: A Simple Demonstration using Matlab," http://www.anilalexander.org/publications/, 2004.

[23] P. Kocher and J. Jaffe and B. Jun, "Differential power analysis," *Advances in Cryptology - CRYPTO*, pp. 388–397, 1999.

VizMIR: A Cross-media Music Retrieval System Supporting Bidirectional Transformation between Mood-based Color Changes and Tonal Changes in Music

Shuichi Kurabayashi and Yoshiyuki Kato
Faculty of Environment and Information Studies, Keio University
5322 Endo, Fujisawa, Kanagawa 252-0882, Japan
{kurabaya, t10247yk }@sfc.keio.ac.jp

Abstract—**VizMIR is a music retrieval system that provides an intuitive user interface to search for music based on the sentiment that it evokes. The unique feature of this system is a cross-media retrieval mechanism that accepts a sequence of images to describe user requirements for mood transitions within a musical composition. VizMIR has a hybrid metric space for converting color change in images to continuous tonal changes in music, and vice versa. When a user enters a sequence of images as a query specifying the desired changes of mood in music, VizMIR measures the color distance in the image sequence, converts the calculated distance into the distance of the movement of musical tonality, and finds music that has the same or similar tonality movement. To support this bidirectional conversion of distance, we design two metric spaces as topologically equivalent structures, and provide a bridge function that maps a distance measured in the color metric space into one in the tonality metric space. This system enables users to search music by subtly manipulating queries through trial and error, and this is easy to use because images are suited to interactive manipulation. This method is useful in searching for music unknown to the user that evinces a mood satisfying the user's preferences.**

Keywords – Cross-Media Retrieval, Emotion-Aware Search, User Interface.

I. INTRODUCTION

In this paper, we describe the *VizMIR* system [1] and its implementation framework using modern web technologies. Our system provides an intuitive user interface to formulate mood-based queries to find music suited to the user's disposition at the time. This system provides a "bridge" between music and images to enable users to search for their preferred songs by using a sequence of images representing the mood expressed by the desired music. Such cross-media retrieval methods are considered very important for designing user-centered music retrieval systems [2].

With the rapid progress of computing technologies, ever more songs are being digitized and stored in online libraries and on personal devices. Due to their proliferation, portable and personal devices, such as tablet computers and smartphones, are commonly used to listen to music. Such proliferation and diversity of digital media increases the demand for an effective music retrieval system [3]. However, it is difficult to find music satisfying our preferences at any given time because the sentiment expressed by a piece of music varies with its progression. Musical data consists of non-verbal elements that proceed along the timeline of the musical composition. The context and temporal transitions of music deeply effect listeners' emotions. In order to find a musical composition the progression of which corresponds to changes in mood desired by the user, the user typically needs to listen to several parts of a piece of music in repositories, such as online music stores and personal music players. Owing to the temporal nature of music, it is difficult to develop an effective music search environment where users can retrieve specific music samples by using intuitive queries. This is because an effective search through a temporal structure requires that the system recognize the changing features of the content in a context-dependent manner.

To retrieve music intuitively, the concept of the "impression" that a musical composition makes on the listener is of great importance. This is because studies have shown that many users consider their feelings and moods to be among the most significant factors motivating them to listen to music. However, in spite of the fact that young users tend to select music according to their disposition, they are frustrated in their attempts to find and retrieve their desired music in a given mood. This is on account of the absence of technology that allows users to enter visual queries to specify the mood of the desired music and the impression that the user wishes for it to create. In particular, in order to find recently released music, or music that may be unknown to the user that is nonetheless appropriate for his/her mood at a given time, a method to effectively communicate the user's desire for music according to a particular disposition is required [3]. Users need a toolkit that assists them to form their own queries using trial and error. Thus, an intuitive user interface (UI) to effectively communicate the demands of users for music is desirable.

With this objective in mind, we propose an impression-aware music retrieval method that offers a cross-media query model using image files as a medium to describe user demands for mood-based music. It is important to develop a stream-oriented query construction method for music because the content of music as well as the impression it creates on the listener change with time. Our query model interprets the effects of temporal changes in media features, such as tonality in music and color in images. This paper presents a prototype system that carries out web-based music

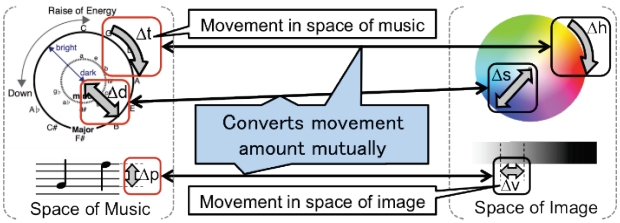


Figure 1. An overview of cross-media retrieval using the delta equation on each media data.

retrieval by considering changes in the mood evinced by the musical composition at hand.

Our design principle for this system is to make it possible for a user to search musical content invisible to him/her by using visible image content. In recent research on synesthesia in psychology, it has become clear that a significant correlation exists between color sense and musical elements [4]. Thus, we think that it is reasonable and beneficial to represent the impression created by musical compositions as sequences of colors in order to design and implement an intuitive user interface for music database systems. This concept is well suited to web-based music retrieval because the web is a visual medium. Thus, a crucial aspect of this system is a cross-media query interpretation method that recognizes how media changes with time by using two metric spaces to calculate the distance between the current and the previous states of the media content in question. For music, we implement a tonality-based metric space. In this tonality space, a song can be modeled as a trajectory in three dimensions. Our system extracts the temporal transition of tonality in a song to analyze emotive transitions occurring with time because tonality is one of the most important factors in determining the overall mood of a musical composition [5][6]. Corresponding to the tonality metric space, we have developed a metric space to compute color distance using the hue, saturation, and value (HSV) [7] color space. HSV is a widely adopted space in image and video retrieval because it describes perceptual and scalable color relationships.

The system transforms invisible changes in the impressions created by music into visible changes in color, and vice versa. Our approach converts a "delta value," which represents distance in each space, between two spaces rather than a feature value itself because the system focuses on how changes of mood in music effect human perception. The most important feature of the two metric spaces is their configuration as topologically equivalent structures (Figure 1). Each axis in the music space has a corresponding axis in the color space. Specifically, tonality is associated with hue, pitch with value, and major/minor with saturation. Thus, a specific distance in music space can be converted into the same distance in color space.

We implement a prototype of our system using HTML5 technologies. The implemented prototype assumes application to online music stores as a front-end user interface. The advantage of this system is an intuitive method for users to edit a query using trial and error depending on their evaluation of the results. The method makes it possible for users to describe changes in impressions created by music, which are difficult to represent directly, as a sequence of images through a visually enhanced user interface, wherein the order of the images represents a change of impression. Thus, our system provides a fundamental framework for implementing the UI of an online music database system.

The remainder of this paper is structured as follows. Section II presents the motivating example of our query processing. Section III briefly summarizes related work in the area. Section IV describes the fundamental concept and the system architecture, whereas we detail our prototype system implementation in Section V. Section VI discusses our feasibility studies, and we offer concluding thoughts in Section VII.

II. MOTIVATING EXAMPLES

In this section, we present two examples motivating our stream-oriented cross-media music retrieval. The first example situation assumes that a user has a lot of music in his/her portable music player and wants to listen to music suited to his/her mood, but does not have an idea of the title and the artist of the type of music in question. In this case, the VizMIR system enables the user to retrieve the desired music by describing moods represented by it using a sequence of images in a trial-and-error method. The user chooses several images from his/her collection of pictures in portable device, and the system find music that creates impressions similar to those created by the selected image sequence.

The second example situation assumes that the user is an illustrator, and wants to make a slideshow of his/her own pieces of illustration in order to seek background music for it. The user has candidates for the slideshow but has no clear idea about the exact composition of the slideshow or the musical piece to serve as background for it. In this case, the user can retrieve music related to changes in the slideshow by revising the order of the candidates, not only for the slideshow but also for forming a query.

III. RELATED WORKS

Conventional music database systems available on Internet use metadata, such as genre and artist name, as indexing keys. As such, fundamental metadata are not sufficient to retrieve music without detailed knowledge of the target data. The music information retrieval (MIR) system is a well-known means of helping users find music by using several intuitive queries [1][8]. However, such approaches cannot be applied to find music that is unknown to the user. Thus, there is a need for a retrieval mechanism for music that users have not heard before [9].

In content-based MIR methods, the system analyzes and extracts several significant features from a musical composition in order to identify equivalent or highly similar music samples in a database. There are several choices of input, such as the user profile-based approach [10], the chord-based approach [11], and the query-by-humming [12][13][14]. Content-based methods are advantageous with regard to ease of input and the ability to generate a large amount of information reflecting the musical content. As

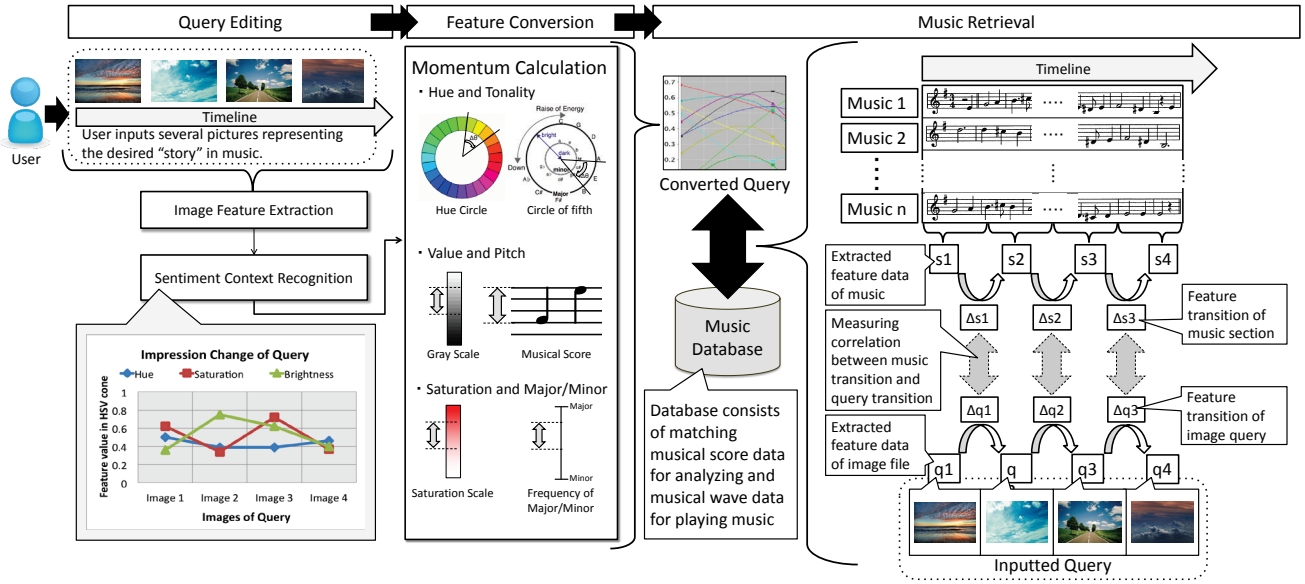


Figure 2. VizMIR system architecture that realizes a visual query construction for retrieving music by converting color changes into tonal changes.

content-based technologies are very effective in retrieving musical equivalents to queries, they are widely used for copyright protection in online music sharing services.

Music visualization systems partially help users find new and unknown music. There are several music visualization systems that utilize the cross-media relationship between color and tonality proposed in [15][16][17][18][19]. An impression-based music visualization method that utilizes the result of a synesthesia study [4] was proposed in [20]. This uses a color sense of tonality to view the harmonic structure of and relationships between important regions in a musical composition. Pampalk et al. [21] proposed an interface for discovering artists by using a ring-like structured visual UI, and Knees et al. [22] developed a method of visually summarizing the contents of music repositories. Stober et al. [23] proposed an interface that can conduct music searches based on ambiguous demands. Research in [24] presents "GlobalMusic2One," a portal site for visualizing songs using a two-dimensional similarity map for explorative browsing and target-oriented finding. Cross-modal media analysis and retrieval methods are proposed in [25][26]. These systems implement user-centered music retrieval by considering user preferences.

It is difficult to create a metric space that can measure temporal changes in heterogeneous media data. In order to design and implement a cross-media retrieval system that considers changes of moods in images and music, it is necessary to develop a new method that transmits feature value bidirectionally between images and music, instead of a knowledge base of music and colors referring to synesthesia.

The most significant difference between conventional approaches and ours is that our system focuses on the development of a method for mood-based cross-media retrieval. Conventional cross-media retrieval methods do not have the capability to detect an impression and a mood in temporal music stream. Our system analyses emotional transitions by capturing the progression of tonality as a function of "how the music sounds." Further, our system allows users to describe their music demands by using an image sequence. Our system is unique in supporting such a cross-media query interpreting mechanism for continuous multimedia data.

IV. System Architecture

Figure 2 shows the fundamental system architecture of VizMIR. The core components of VizMIR constitute four data structures and three functions. The system models the concept of "change in sentiment" by measuring the distance between successive temporal transitions in the media data. Our method succeeds in cross-media retrieval by comparing the results of continual sentiment analysis of music and images. This system provides two delta functions to analyze temporal changes in the media data and to generate sequential values representing changes of mood in them. The system calculates the sentiment-oriented relevance score of music and images by comparing the calculated delta values.

We show an example query in Figure 3. This query represents changes in impression as follows: the brightness (value in the HSV color model) gradually increases and then decreases; the hue (type of color) gradually changes from blue to red; and the saturation (vividness of colors) drastically increases in the middle of the query. The system translates these features of the query into musical features as follows: the pitch gradually increases and then decreases, the tonality gradually changes, and the major/minor changes drastically. The system retrieves music by calculating the relevance score of the query translated into music.

A. Architectural Overview

As shown in Figure 2, the system consists of three main components: 1) a query editor, 2) a feature conversion module, and 3) a retrieval engine.

The query editor is the front-end module of the system. This module provides a set of operations to prepare and

modify image files as a query. For example, the system implements an image-editing operator equipped with several color filters to change the overall impression of the image.

The feature conversion module provides fundamental data conversion functions applied to musical instrument digital interface (MIDI) data and bitmap image data. The feature conversion module generates metadata vectors from the image and MIDI files.

The retrieval engine calculates how the media data change with time by applying distance functions to the generated metadata. The system provides a bridging mechanism between the musical tonality metric space and the HSV color metric space. The bridge converts a distance calculated in the music space into a distance in the image space in order to retain the impression factor from one medium to the other. For example, in the distance conversion mechanism, hue, which is a type of color, corresponds to tonality, which is a type of musical structure. By converting distances between heterogeneous metric spaces, the system realizes cross-media retrieval for stream media such as music. Finally, the retrieval engine compares the set of distance values to calculate the relevance of the music to the image query. In the following sections, we describe in detail the fundamental data structures and functions involved.

B. Image Sequence as a Query

VizMIR accepts an image sequence as a query that represents how the media data changes in mood with time. An image sequence object Q is defined as follows:

$$Q := \langle\langle h_1, s_1, v_1\rangle \ldots \langle h_i, s_i, v_i\rangle\rangle \quad (1)$$

where h_i is the hue data, s_i is the saturation data, and v_i is brightness data in the i-th image of the query. The system converts RGB color values of each image into HSV triples at the pixel level. We adopt the following well-known RGB-to-HSV conversion equation.

$$\mathrm{V} = \max(\mathrm{R}, \mathrm{G}, \mathrm{B}) \quad (2)$$

$$\mathrm{S} = 255 \times \frac{\max(\mathrm{R}, \mathrm{G}, \mathrm{B}) - \min(\mathrm{R}, \mathrm{G}, \mathrm{B})}{\max(\mathrm{R}, \mathrm{G}, \mathrm{B})} \quad (3)$$

$$\mathrm{H} = \begin{cases} 60\dfrac{B-G}{\max(\mathrm{R},\mathrm{G},\mathrm{B}) - \min(\mathrm{R},\mathrm{G},\mathrm{B})} & R == \max(\mathrm{R},\mathrm{G},\mathrm{B}) \\ 60\left(2 + \dfrac{R-B}{\max(\mathrm{R},\mathrm{G},\mathrm{B}) - \min(\mathrm{R},\mathrm{G},\mathrm{B})}\right) & G == \max(\mathrm{R},\mathrm{G},\mathrm{B}) \\ 60\left(4 + \dfrac{G-R}{\max(\mathrm{R},\mathrm{G},\mathrm{B}) - \min(\mathrm{R},\mathrm{G},\mathrm{B})}\right) & B == \max(\mathrm{R},\mathrm{G},\mathrm{B}) \end{cases} \quad (4)$$

We define the metric space of images as the HSV color metric space with three axes: hue, saturation, and value. These three elements are significant factors affecting the impression of the image. Hue represents the differences of color phases, such as red, yellow, green, and blue. In the HSV cone of images, the hue is represented by an angle. The system converts the extracted hue angle in the HSV cone of images into a hue scalar h, which is a value between 0 and 1. Saturation is the vividness of color. In our system, saturation is an average value of the vividness in an image. The system processes this vividness value of an image into a saturation scalar s, which is a value between 0 and 1. Here, 0 and 1 represent the lowest and the highest value, respectively. Value is the brightness of color. Our system calculates the value (brightness) as an average value of color brightness in an image, processes the brightness value of an image into a brightness scalar v, which is also between 0 and 1.

Figure 3. An example impression query consisting of four images

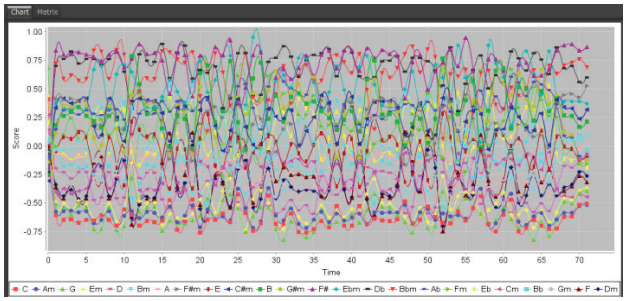


Figure 4. A visualization of tonality change in one music item. The tonality changes with time.

C. MIDI Song Data

Our system uses standardized MIDI data format as the primary data format for storing music. MIDI stores note-on signals and corresponding note-off signals sequentially because it was developed in order to automate keyboard instruments. The system represents a MIDI file F:={$n_1(t, p, d)$, n_2, …, n_k}, where n_i represents the i-th note whose attributes are 1) t: the start time of the note, 2) p: the pitch of the note, and 3) d: the duration of the note. F is a sequential set of k-tuple data.

Our system provides a matrix structure that represents the continuous variation in and distribution of pitch in the target music data. We call the data structure a music pitch matrix. The pitch matrix is a 128 × n matrix, which is given as the data matrix. MIDI specifications define the domain of pitch value between 0 and 127. A musical composition is expressed as a set of m timelines. Each timeline is characterized by a note on information for 0 to 127 pitch level. When the 12th note is on the m-th section, $c_{[12,m]}$ is 1. The pitch matrix P is defined as follows:

$$\mathrm{P} := \begin{pmatrix} c_{[0,0]} & \cdots & c_{[0,n]} \\ \vdots & \ddots & \vdots \\ c_{[m,0]} & \cdots & c_{[m,n]} \end{pmatrix} \quad (5)$$

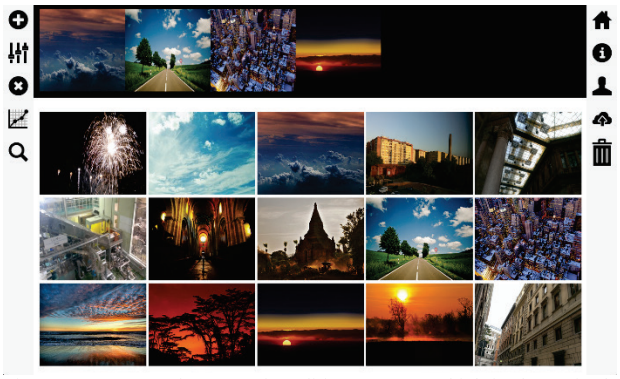


Figure 5. An example screen for editing a query, which is shown in the black area at the top, by utilizing photo stocks shown at the bottom.

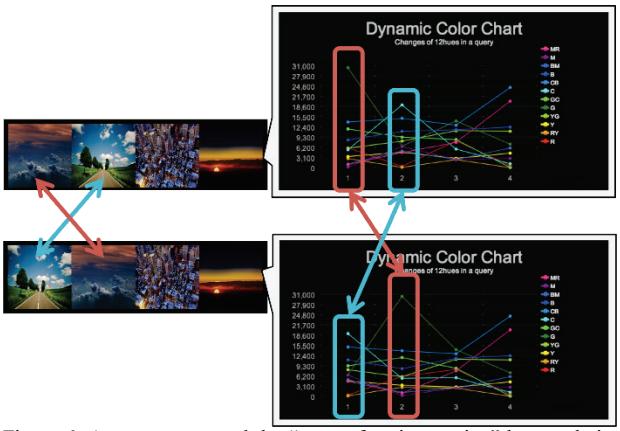


Figure 6. A user can control the "story of an impression" by reordering images. In this case, the impression in the blue rectangle is moved to the head of the story.

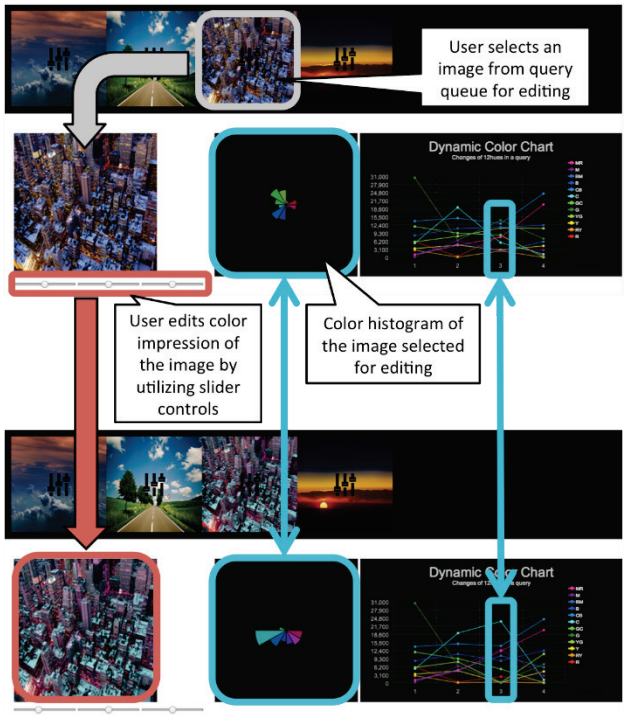


Figure 7. A user can directly apply color filters to images for controlling the impression in a fine-grained way.

where $c_{[i,j]}$ denotes the j-th pitch status at the i-th time duration. We have implemented the MIDI analysis modules for converting MIDI into a musical score-like data structures by using our MediaMatrix system [27], a stream-oriented database management system.

D. Music Data

When VizMIR receives a query consisting of several images, the system divides every song in its database into sections, such that the number of sections is equal to the number of images entered as image query. A music object M is defined as follows:

$$M := \langle\langle t_1, d_1, p_1\rangle \ldots \langle t_i, d_i, p_i\rangle\rangle \quad (6)$$

where t_i is the tonality data, d_i is the deviation data, and p_i is the pitch data, in the i-th section of the music. We define the metric space of the music using three axes: tonality, pitch, and major/minor. These three elements are significant factors effecting the impression of music.

Tonality is the structure of music and is composed of sequential musical notes. There are 24 tones consisting of 12 major and 12 minor tones. As shown in Figure 3, tonality in music changes with time and this causes changes in the impression of the music. Figure 3 shows movements of 24 types of tonality in a song using our tonality analysis function implemented in [27]. In musical theory, a circle of fifths defines the difference or similarity between each pair of the 24 tonalities [5][6]. Each tonality can be represented by an angular value on the circle of fifths. The system processes this angular value into a tonality scalar t, which is a value between 0 and 1. Thus, the system converts the distance measured as the angle of the hue of the image into a scalar quantity representing the angle of the tonality.

Major/minor refers to the deviation of tonality within a music section. The system calculates the deviation of tonality in a music section, and converts the deviation value into a major/minor scalar d. This has a value between 0 and 1, representing the maximum minor deviation and the maximum major deviation, respectively.

Pitch is a value of pitch in a musical score. The system calculates the average of the pitches in a music section, and converts each of the average values into a scalar quantity p. Values of p also fall between 0 and 1, which quantities represent the lowest and the highest pitch, respectively.

E. Image Distance Calculation Function

We design the following three functions in order to calculate a distance in an image sequence query Q:

- The distance in hue between the i-th image and the (i+1)-th image is $\Delta_{hi} := |h_i - h_{i+1}|$, where h is the hue angle in the HSV cone.
- The distance in saturation between the i-th image and the (i+1)-th image is $\Delta_{si} := |s_i - s_{i+1}|$, where s is the saturation coordinate in the HSV cone.
- The distance in value between the i-th image and the (i+1)-th image is $\Delta_{vi} := |v_i - v_{i+1}|$, where v is the value coordinate in the HSV cone.

F. Tonality Distance Calculation Function

We design the following three functions in order to calculate a distance in a music object *M*:

- The distance in tonality between the *i*-th section and the (*i*+1)-th section is $\Delta_{ti} := |t_i - t_{i+1}|$, where *t* is the tonality angle in the circle of fifths.
- The distance in tonality deviation between the *i*-th section and the (*i*+1)-th section is $\Delta_{di} := |d_i - d_{i+1}|$, where *d* is the deviation in tonality.
- The distance in pitch between the *i*-th section and the (*i*+1)-th section is $\Delta_{pi} := |p_i - p_{i+1}|$, where *p* is the pitch.

G. Cross-Media Relevance Calculation Function

The system provides a function to calculate the relevance of the music to the query. The function is defined as follows: $f(a,b) \rightarrow 1 - |a - b|$, where *a* and *b* form a pair of distance changes according to the dual-metrics relation. The system calculates a correlation value for each pair of metrics using this function. The relevance of the music to the image query is represented as follows:

$$\gamma(\Delta q, \Delta m) := \frac{\sum_{i=1}^{n} \frac{s(\Delta_{hi}, \Delta_{ti}) + s(\Delta_{hi}, \Delta_{ti}) + s(\Delta_{hi}, \Delta_{ti})}{3}}{n} \quad (7)$$

where *n* is the number of images entered as part of the image sequence object *M*, as well as the number of divided music sections.

V. Prototype System Implementation

We have implemented a prototype of the proposed system. The screenshots of the prototype, which uses HTML5 Canvas and JavaScript, are shown in Figure 5 – Figure 8. The system implemented consists of three modules: the query editor, the feature conversion module, and the music retrieval engine.

The query editor is the main user interface shown in Figure 5. Users can form a query by selecting four images and by revising the appearance of the query. In Figure 5, a user has selected four images as elements of a query. The system provides two ways for the user to edit the query:

- The first manner is to edit the entire impression of the query by revising the order of the images as shown in Figure 6. The system allows the user to intuitively perform this using a drag-and-drop operation.
- The second manner is to edit the partial impression of the query by changing the color of an image, as shown in Figure 7. The system allows the user to do this by moving the three sliders associated with each of the HSV values.

When the user finishes editing the query, he/she submits the images for the music search. The feature conversion module extracts the semantic color movement of the submitted images and generates a query consisting of a virtual musical feature in order to retrieve music. Finally, the music retrieval engine calculates the relevance of the candidate music with

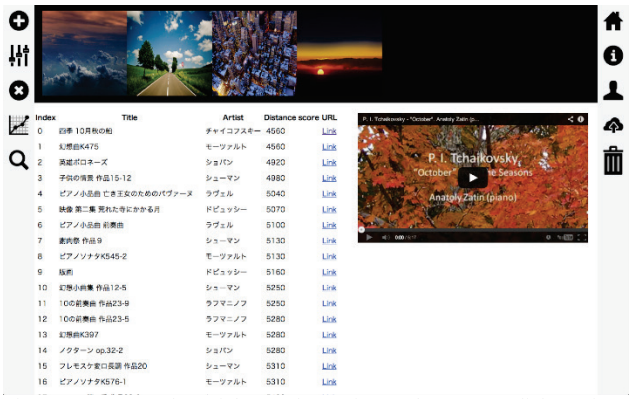


Figure 8. A screenshot of the result set view. When a user clicks an item in the result set, the system opens a video playback screen and plays a video corresponding to the selected MIDI data.

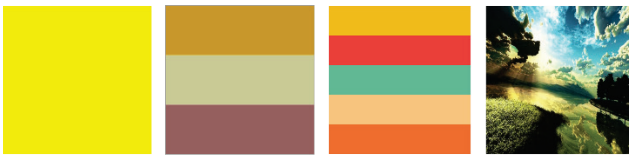


Figure 9. Examples of query components.

the generated query, and shows indexed music as a result according to the calculated relevance score, as shown in Figure 8. If the result does not satisfy the preferences of the user, the user revises the query using the query editor. By repeating the above process, the user searches music through trial and error.

The system has a backend MIDI analyzer. When a user enters a query, the system invokes the MIDI file analyzer with a section number parameter. The system analyzes the MIDI files in an on-the-fly manner in order to extract the tonality transition according to the section number parameter passed as a component of a query. We have implemented the MIDI file analyzer using HTML5 FileReader object and ArrayBuffer object. When finished with the analysis process, the MIDI file analyzer encodes the analysis result into JavaScript Object Notation (JSON) format and passes it to the distance calculation module. This procedure allows our system to share the JSON-encoded figure among multiple web workers to parallelize the distance calculation.

The relevance calculation module compares the queries and the database contents. This retrieval process is parallelized by the web workers application programming interface (API), and the retrieved songs are presented to the user through the search result visualization engine. This system spawns real operating system (OS)-level threads from the web workers API to parallelize the retrieval process. Modern HTML5 technologies enable us to implement complex processes in web browsers. Figure 8 shows a screenshot of an example of the result set and its preview screen. When a user clicks an item in the retrieval results, the system opens a video playback screen and plays a video corresponding to the selected MIDI data. In this case, our system assumes that the MIDI data is a fundamental metadata for a song. Thus, our system stores both a MIDI

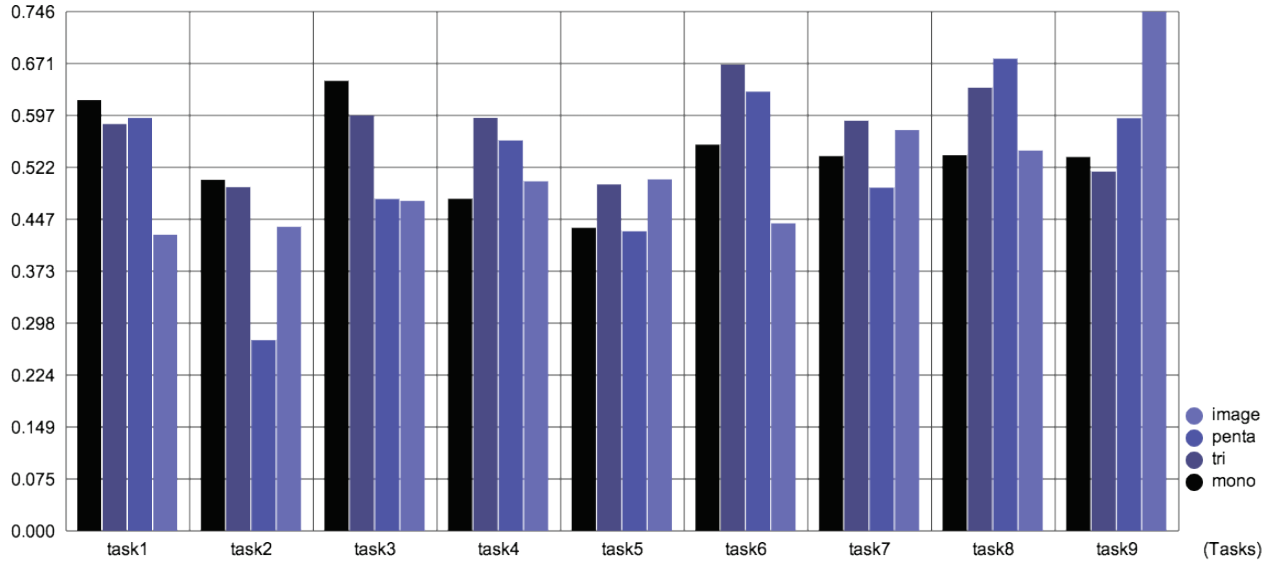


Figure 10. NDCG scores calculated by summing up all subjects, including novices and experts. Simple queries such as mono-color and tri-color images are suitable for simple tasks, and complex queries such as penta-color images and natural images have achieved better results in complex tasks.

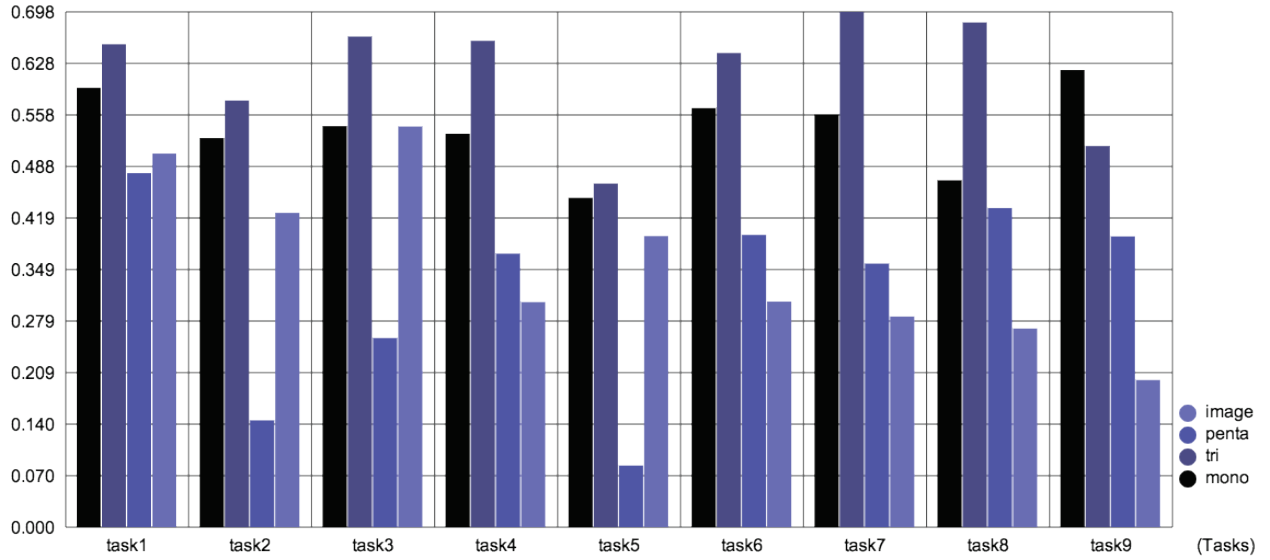


Figure 11. NDCG scores of nine tasks performed by novices. Novices achieved better scores when they used simple queries, such as ones involving mono-color and tri-color images.

file and a corresponding raw media file, such as a video file and an audio file.

VI. EXPERIMENTAL STUDIES

This section details several experiments to evaluate the effectiveness of our VizMIR system when applied to existing 100 western classic music. We prepared for our subjects nine music search tasks that are categorized into three difficulty levels: EASY, NORMAL, and HARD. Each level contains three tasks. The tasks are shown in Figure 9. Table I shows the details of the impression transition of each task in increasing order of difficulty. As can be seen, task 4 is more difficult than task 1, and task 9 is more difficult than task 4. The tasks represent the required impression from the desired music using a combination of 10 kinds of feature words associated with tonality. Subjects form queries by selecting four images according to the requirements of the task and situation at hand. Task 1 requires that the subjects find music that consists of “soft” impressions followed by “gorgeous” impressions. “Soft(C, Db)” means that the impression “soft” corresponds to tonalities “C” and “Db”. In this table, “b” denotes “flat”, and “m” denotes minor tonality.

We asked 24 subjects (13 female and 11 male) to search for music through our system by using the following types of queries:

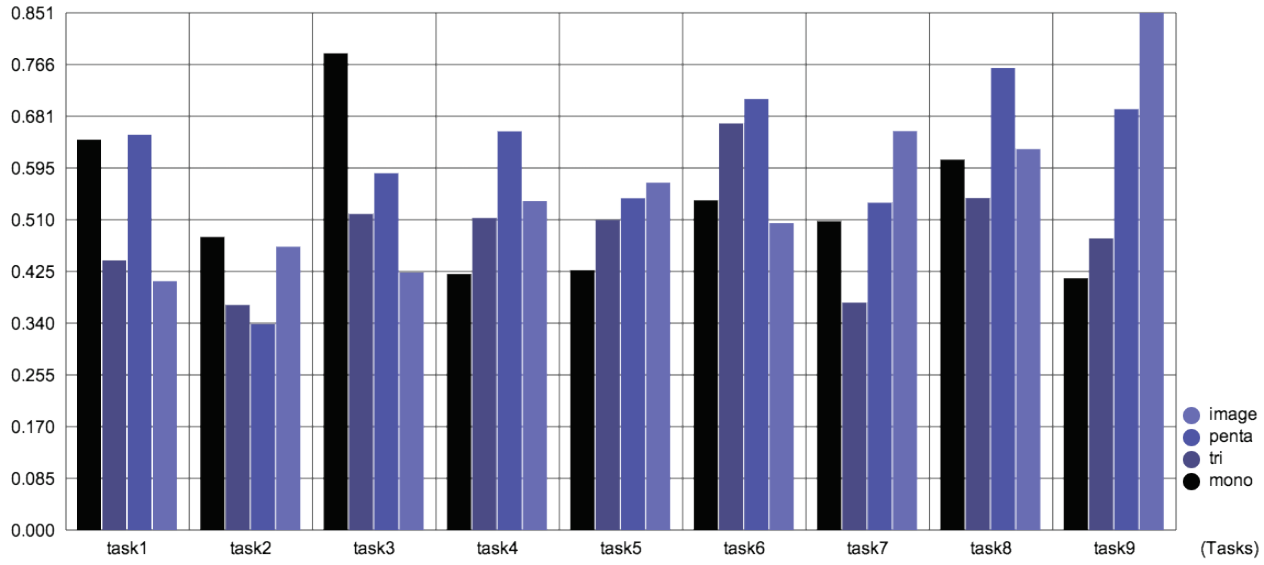


Figure 12. NDCG scores of nine tasks performed by experts. In the case of expert users, queries using full color and penta-color images are more effective when applied to more complex tasks.

- "image" query: the subjects construct a query by using full-color pictures.
- "penta" query: the subjects construct a query by using five-color images.
- "tri" query: the subjects construct a query by using three-color images.
- "mono" query: the subjects construct a query by using a single color image.

TABLE I. THREE-LEVEL SEARCH TASKS

Level	ID	Impression Transition
EASY	1	Soft(C, Db) → Gorgeous(D, Eb)
	2	Mysterious(F#m, Gm) → Solitary(Abm, Am, Abm)
	3	Calm(G, Ab, A) → Plaintive(Bm, Cm)
NORMAL	4	Calm(G, Ab, A) → Powerful(C, Db) → Gorgeous(D, Eb)
	5	Solitary(Abm, Am, Abm) → Sorrowful(Em, Fm) → Weird(Dbm, Dm, Ebm)
	6	Mysterious(F#m, Gm) → Calm(G, Ab, A) → Fresh(B, Bb)
HARD	7	Fresh(B, Bb) → Powerful(C, Db) → Gorgeous(D, Eb) → Soft(C, Db)
	8	Plaintive(Bm, Cm) → Solitary(Abm, Am, Abm) → Sorrowful(Em, Fm) → Plaintive(Bm, Cm)
	9	Soft(C, Db) → Solitary(Abm, Am, Abm) → Calm(G, Ab, A) → Mysterious(F#m, Gm)

Example images used in this experiment are shown in Figure 9. Thus, the subjects translate search tasks into image sequences while satisfying the above constraints.

To evaluate this experiment, we computed the normalized discounted cumulative gain (NDCG) as follows:

$$DCG = \sum_{i=1}^{5} \frac{rel_i}{log_2 i} \quad (8)$$

$$IDCG = \sum_{i=1}^{5} \frac{rel'_i}{log_2 i} \quad (9)$$

$$NDCG = \frac{DCG}{IDCG} \quad (10)$$

where rel_i is the average of survey scores from the test subjects in order of calculated distance, and rel'_i represents the average of the scores in descending order. We have created the correct set by comparing the tonality description in Table I with the automatically extracted tonality metadata from the data set containing 100 music items. Figure 10, Figure 11, and Figure 12 show the NDCG of our system when applied to tasks 1–9. A higher score implies a better retrieval precision. We divided the results according to the musical background of the subjects: novices who have never played music, and experts skilled at playing music.

Figure 10 shows the NDCG of all results of music retrieval for the nine tasks. Overall, simple queries such as those involving mono-color and tri-color images are suitable for simple tasks, and complex queries such as those involving penta-color images and natural images achieved better results in complex tasks. Figure 11 shows the NDCG of music retrieval by novices. It appears that novices achieved better scores when they used simple queries, such as ones using mono-color and tri-color images. Figure 12 shows the NDCG of music retrieval by experts. For these users, full color images and penta-color queries are more effective when applied to more complex tasks. The most important result is the difference in distribution of NDCG scores between the novices and the experts. Novices retrieved music effectively by using three-color images, regardless of the complexity of the task. On the other hand, as shown in Figure 12, experts retrieved music effectively by using full color images when the complexity of a task was

high (e.g., single color images are suitable for a simple task such as task 3, and full color images are suitable for a complex task such as task 9).

These results imply that skill at playing music effects cross-modal sensibility for images and music. Experts can use their musical sensibilities to form queries by using images, and novices find it difficult to detect musical impressions as images.

VII. CONCLUSION

In this paper, we proposed the *VizMIR* system, a cross-media retrieval system for music that can provide an intuitive visual retrieval method. The unique feature of this system lies in its construction of image-based queries to represent the transition in mood within a musical composition. VizMIR has a hybrid metric space for converting color change of images to continuous tonal change of music, and vice versa. We implemented the prototype system by utilizing HTML5 technologies. This implementation system supports the on-the-fly image uploading and configuring in order to create a query. We performed an evaluation of our system using a database of classical music. Experimental results showed that our visually-enriched query model performs well in practice. In the future, we plan to develop a personalized query interpretation and a social-network-based query recommendation system by building on this approach.

REFERENCES

[1] Kato, Y., and Kurabayashi, S., "Cross-media retrieval for music by analyzing changes of mood with delta function for detecting impressive behaviours," In Proceedings of the Eighth International Conference on Internet and Web Applications and Services (ICIW 2013), pp. 236-239, 2013.

[2] Liem, C., Esk, D., and Tzanetskis, G., "The need for music information retrieval with user-centered and multimodal strategies," In Proc. Of the 1st International ACM Workshop on Music Information Retrieval with User-Centered and Multimodal Strategies (MIRUM), pp. 1-6, 2011.

[3] Goto, M., and Hirata, K., "Recent studies on music information processing," Acoust. Sci. Technol., vol. 25, no. 6, pp. 419-425, 2004.

[4] Peacock, K., "Synesthetic perception: alexander scriabin's color hearing," Music Percep. vol. 2, no. 4, pp. 483–506, 1985.

[5] Temperley, D., "Music and probability," Cambridge, MA: MIT Press, 2007.

[6] Krumhansl, C. L., "Cognitive foundations of musical pitch," New York, NY: Oxford Univ. Press, 1990.

[7] Smith, A. R., "Color gamut transform pairs," In Proc. of the 5th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '78), pp.12-19, 1978.

[8] Typke, R., Wiering, F., and Veltkamp, R., "A survey of music information retrieval systems," In Proc. the 6th International Conference on Music Information Retrieval (ISMIR 2005), Univ. of London, 2005, pp. 153-160.

[9] Kuo, F. F. and Shan, M. K., "Looking for new, not known music only: music retrieval by melody style," In Proc. Of the 4th ACM/IEEE-CS Joint Conf. Digital Libraries, (JCDL '04), ACM Press, 2004, pp. 243-251.

[10] Hijikata, Y., Iwahama, K., and Nishida, S., "Content-based music filtering system with editable user profile," In Proc. of the 2006 ACM Symposium on Applied Computing, pp. 1050-1057, 2006.

[11] Bello, J.P., "Audio-based cover song retrieval using approximate chord sequences: testing shifts, gaps, swaps and beats," In Proc. the 8th International Conference on Music Information Retrieval (ISMIR 2007), pp. 239-244, 2007.

[12] Ghias, A., Logan, J., Chamberlin, D., and Smith, B.C., "Query by humming: musical information retrieval in an audio database," In Proc. ACM Multimedia 95, pp. 231–236, 1995.

[13] Dannenberg, R.B., Birmingham, W.P., Tzanetakis, G., Meek, C., Hu, N., and Pardo, B., "The MUSART Testbed for Query-by-Humming Evaluation," In Proc. of the 4th international conference on music information retrieval (ISMIR 2003), pp. 34-48, 2003.

[14] Shifrin, J., Pardo, B., Meek, C., and Birmingham, W., "HMM-based musical query retrieval," In Proc. of the 2nd ACM/IEEE-CS joint conference on digital libraries (JCDL 2002), pp. 295- 300, 2002.

[15] Craig, S., "Harmonic visualizations of tonal music," In Proc. of the International Computer Music Conference (ICMC 2001), MPublishing, University of Michigan Library, pp. 423-430, 2001.

[16] Gómez, E. and Bonada, J., "Tonality visualization of polyphonic audio," In Proc. of the International Computer Music Conference (ICMC 2005), MPublishing, University of Michigan Library, 2005.

[17] Mardirossian, A. and Chew, E., "Visualizing music: tonal progressions and distributions," In Proc. of the 8th International Conference on Music Information Retrieval (ISMIR2007), pp. 189-194, 2007.

[18] Ciuha, P., Klemenc, B., and Solina, F., "Visualization of concurrent tones in music with colours", n Proc. of the 18th International Conference on Multimedia 2010 (MM '10), pp. 1677- 1680, ACM, 2010.

[19] Cooper, M., Foote, J., Pampalk, E., Tzanetakis, G., "Visualization in audio-based music information retrieval," Computer Music Journal, Vol. 30, No. 2, pp. 42-62, MIT Press, 2006.

[20] Imai, S., Kurabayashi, S., and Kiyoki, Y., "A music database System with Content analysis and visualization mechanisms," In Proc. of the IASTED International Symposium on Distributed and Intelligent Multimedia Systems, ACTA Press, pp. 455-460, 2008.

[21] Pampalk, E. and Goto, M., "Musicrainbow: a new user interface to discover artists using audio-based similarity and web-based labeling," In Proc. of the 7th International Conference on Music Information Retrieval (ISMIR 2006), pp. 367-370, 2006.

[22] Knees, P., Schedl, M., Pohle, T., and Widmer, G., "An innovative three-dimensional user interface for exploring music collections enriched with meta-information from the web", In Proc. of the 14th ACM International Conference on Multimedia (MM '06), pp. 17-24, 2006.

[23] Stober, S. and Nürnberger, A., "MusicGalaxy: A multi-focus zoomable interface for multi-facet exploration of music collections", In Proc. of the 7th International Symposium on Computer Music Modeling and Retrieval (CMMR 2010), pp. 259–272, Springer, 2010.

[24] Dittmar, C., Großmann, H., Cano, E., Grollmisch, S., Lukashevich, H., and Abeßer, J., "Songs2See and GlobalMusic2One: two applied research projects in music information retrieval at Fraunhofer IDMT," In Proc. of the 7th International Symposium on Computer Music Modeling and Retrieval (CMMR 2010), pp. 259–272, Springer, 2010.

[25] Mao, X. and Lin, B., "Parallel field alignment for cross media retrieval categories and subject descriptors," In Proc. of the 21st ACM Conference on Multimedia (MM '13), pp. 897-906, ACM Press, 2013.

[26] Zhang, H., Zhuang, Y., and Wu, F., "Cross-modal correlation learning for clustering on image-audio dataset," In Proc. of the 15th ACM Conference on Multimedia (MM '07), pp. 273-276, 2007.

[27] Kurabayashi, S. and Kiyoki, Y., "MediaMatrix: a video stream retrieval system with mechanisms for mining contexts of query examples", In Proc. of the 15th International Conference on Database Systems for Advanced Applications (DASFAA2010), pp. 452-455, 2010.

The Data Checking Engine: Complex Rules for Data Quality Monitoring

Felix Heine, Carsten Kleiner, Arne Koschel
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science, Hannover, Germany
Email: firstname.lastname@hs-hannover.de

Jörg Westermayer
SHS Viveon
Germany
Email: joerg.westermayer@shs-viveon.de

***Abstract*—In the context of data warehousing and business intelligence, data quality is of utmost importance. However, many mid-size data warehouse (DWH) projects do not implement a proper data quality process due to huge up-front investments. Nevertheless, assessing and monitoring data quality is necessary to establish confidence in the DWH data. In this paper, we describe a data quality monitoring system: The "Data Checking Engine" (DCE). The goal of the system is to provide DWH projects with an easy and quickly deployable solution to assess data quality while still providing highest flexibility in the definition of the assessment rules. It allows to express complex quality rules and implements a two-staged template mechanism to facilitate the deployment of large numbers of similar rules. While the rules themselves are SQL statements the tool guides the data quality manager through the process of creating rule templates and rules so that it is rather easy for him to create large sets of quality rules. The rule definition language is illustrated in this paper and we also demonstrate the very flexible capabilities of the DCE by presenting examples of advanced data quality rules and how they can be implemented in the DCE. The usefulness of the DCE has been proven in practical implementations at different clients of SHS Viveon. An impression of the actual implementations of the system is given in terms of the system architecture and GUI screenshots in this paper.**

***Keywords*—Data Quality, Quality Rules, Data Analysis, Data Quality Monitoring, Data Warehouses**

I. Introduction

Data quality (DQ) is of utmost importance for a successful data warehouse project. In this context, continuous monitoring is an integral part of any DQ initiative. In this paper, we describe a data quality monitoring system called *Data Checking Engine* (DCE) developed collaboratively at the University of Applied Sciences & Arts Hannover and SHS Viveon. The main goal is to provide a flexible, yet simple tool to monitor data quality in DWH projects, which can also be used during the DWH development to test its Extract Transform Load (ETL) process. Implementations of the system have already been used at some key pilot customers of SHS Viveon and continuous improvements of the technical as well as the conceptual parts of the system are based on feedback from those customers gathered during daily usage of the system.

Data rules are used in order to constantly monitor the quality of data of a database. For the definition of these rules, a flexible language is necessary. Quality rules are either derived from business rules or found via profiling or data mining. They are executed either in regular intervals or based on specific events like the completion of an ETL job. The results of checking the rules are recorded in a result repository, which also keeps historical data so that users can evaluate the quality of data over time. As rules will evolve over time, it is necessary to keep a history of rule definitions so that historic results can be related to the correct version of the rule's definition.

We believe that the ability to express complex rules is crucial. A set of hard-coded rule types found in some data quality tools is typically only suitable to detect rather simple quality problems on the attribute or single tuple level. However, there are more complex data quality problems, which cannot be detected using such rules. As an example, consider an error located in the logic of an ETL process. Due to this error, the process fails to reference the correct product group for some of the records of a sales fact cube. The bug is subtle and does not show up very often. At the attribute level all sales records are correct. However, the trend of the time series showing the sales sum with respect to individual product groups will indicate a quality problem. Other advanced data quality problems and according check rules will be explained in sec. IV, which is also one of the major extensions of this article in comparison to [1].

It requires skilled users to write such rules, but larger sets of rules will look similar in structure. They differ only in the tables and attributes they are applied to. Therefore, a template mechanism is useful to help users define such rules. The idea is that only the template creator has to cope with the full complexity; template users can then apply these templates to their tables and attributes.

To avoid discontinuity of the reporting environment for DWH users, re-using existing Business Intelligence (BI) tools is superior over building a specialized quality reporting GUI. Still, it is sufficient to export rule results to a quality data mart, which can then be accessed by any standard BI tool. However, the plain rule results have to be aggregated to more comprehensive quality metrics in a flexible and user defined way.

Furthermore, the rules themselves have to be tested in the development environment before deployment. Thus, an automated transfer and synchronization with the production system is necessary.

In a nutshell, we target the following requirements:

- Express complex rules
- Reduce complexity of rules for end users (by utilizing a template mechanism)

- Execute the rules regularly or upon specific events
- Keep a history of rule definitions and execution results
- Store this history in a quality data mart persistently
- Aggregate the rule results to quality metrics
- Provide export/import mechanism for rule meta data

This paper is an extended version of the paper [1]. The example section has been included to describe new quality rule types and to underline the flexibility of our approach, and the related work section has been revised and extended significantly as well.

The remainder of this paper is organized as follows: In the following section, we give an overview of related work. Section III focuses on the definition of quality rules and explains our template mechanism in general, whereas Section IV illustrates the rule definition language in detail by discussing how to implement frequently occurring sample rules. This section is the major extension of this article in comparison to the earlier version [1]. Section V describes the DCE architecture and in the subsequent section we briefly elaborate on quality metrics. Finally, Section VII (which is also an addition in comparison to [1]) illustrates the DCE concept and GUI in more detail, before we summarize our achievements and give an outlook to our future plans in the final section.

II. Related Work

Over the last decade, much research in the data quality domain has been conducted, see for example [2], [3], [4], or [5]. Research areas related to data quality are outlier detection, data deduplication, data quality monitoring, data cleansing, and data mining to detect quality rules. We are specifically interested in monitoring and reporting data quality, and in algorithms to detect quality rules automatically from existing data. In general, we follow the approach of Kimball [6] who outlines an approach to DQ assessment in DWH systems.

For our work, ideas and formalisms to describe quality rules are highly relevant. Many types of quality rules stem from the field of database constraints, as described by Bertossi and Bravo in their survey [7]. As classical constraints are not very flexible, numerous new kinds of constraints have been proposed in the literature. In [3], Fan describes multiple formalisms that target specific quality problems. *Conditional functional dependencies* are used to express more complex rules spanning multiple tuples of a relation (see also [8], [9]), while *conditional inclusion dependencies* are generalizations of referential integrity checks. The classical *edit rules* of Fellegi and Holt are concerned with the integrity of individual records [10]. In [11], Fan defines *editing rules* to match records with master data. Further rule types include *differential dependencies* [12], *multidimensional conditional function dependencies* [13], and *probabilistic, approximate constraints* [14]. From our point of view, these are examples of specific types of rules. We aim to provide a framework that is able to express any of these rules. As all these approaches can be reformulated to SQL, the DCE is able to execute these rules. However, this leads to rather complicated rule definitions. To make life easier for the end users, we further provide a template approach. With this approach, we can define a template for each of the rule types, so that rules can be instantiated in an easy way.

In the domain of data deduplication (also called record linkage), rules are important to describe matching criteria. As an example, the IntelliClean [15] system uses rules like *<if> condition <then> action with probability p* to match duplicates. Fan et al. [16] introduce *matching dependencies* to describe criteria that are used to identify duplicate records. For a survey of duplicate record detection, see [17].

Another approach is to extend SQL to incorporate data quality features. An example is the FraQL [18] language that specifies pivoting features and allows to integrate user defined grouping and aggregate functions that allow to analyze data more comfortably. The drawback is that a special execution engine is required. Thus, the features of existing relational optimizers are not available or have to be reproduced.

Furthermore, many prototypic research systems and commercial tools are present. For an overview, see [19]. Most existing tools focus on dimension data only and thus stress single record problems and deduplication. The profiling component of existing data quality tools currently provides only basic algorithms to detect quality rules, see [20]. We think that more advanced profiling techniques are necessary to detect quality rules automatically. Basic approaches are found in the domains of data mining and outlier detection, also called anomaly detection. An overview can be found in [21] as well as in the recent book of Aggarwal [22]. We plan to integrate these concepts in a later version of the DCE. For this, we are especially interested in finding outliers in time series (see, e.g., [23], [24]) and algorithms to analyze multidimensional data (see, e.g., [25], [26], or [27]). However, to the best of our knowledge, no tool provides a similar mechanism that allows to build complex rule templates, which can, for example, be used to test indicator values against time series models.

III. Rule Definition Language

A central issue is the language to define the quality rules. On the one hand, it has to be expressive to allow complex rules like time series tests. On the other hand, fast definitions of simple rules like NULL value checks has to be possible. Also, the rule execution is typically critical with respect to execution time and resource consumption. As large datasets have to be checked, an efficient rule execution engine is needed.

Thus, we decided to rely on the native SQL executor of the DBMS. This means, the core of each rule is an SQL statement, which collects the required information from the underlying tables. This statement is written by the DCE user, allowing even vendor-specific optimizations like optimizer hints.

DCE defines a *standard attribute set* for the result tuples. The rule statements have to adhere to this standard. Each statement computes a *result value*, which is the basis for the rule check. For a NULL rule, the result value might be the percentage of NULL values of the checked values. There might either be a single result value or multiple values, broken down by dimensional hierarchies. The latter case might for example yield a percentage of NULL values for each product group in each region. Furthermore, two *base values* can be returned. They can provide additional information for the rule outcome. This might be helpful when interpreting the rule results.

For each rule, *multiple bounds* can be defined, specifying valid ranges for the observed values. The bounds can be

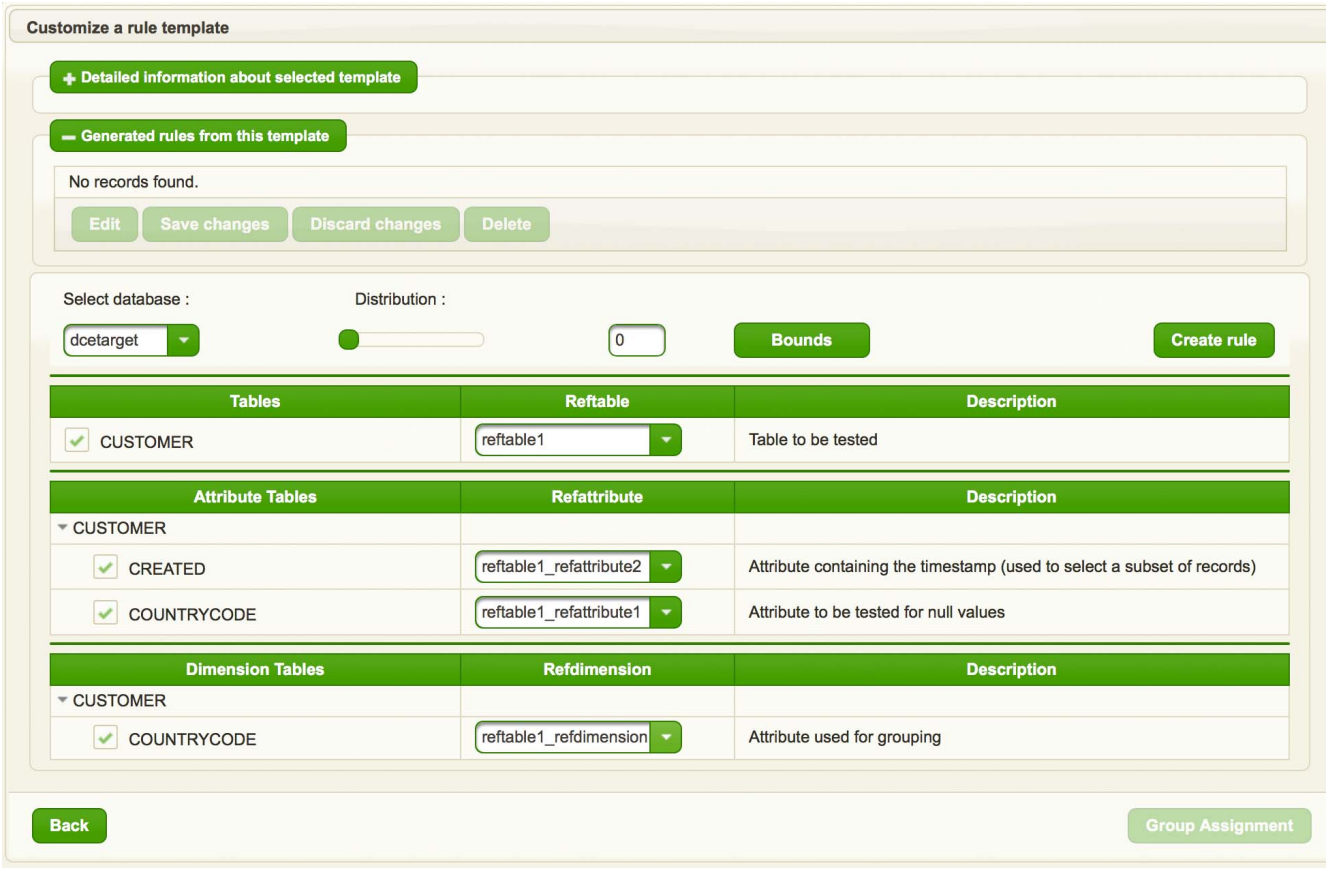


Fig. 1. Instantiating a template

activated or deactivated with respect to all values contained in the result tuple, including both base values and the dimension values. In this way, the bound for NULL values can be normally defined to be 5 percent, however, for specific product groups it might be higher. A specific application for this feature is to change bounds for business metrics, e.g., according to the week day. Typically, the revenue sum for traditional stores might be zero on Sundays.

A *severity* can be assigned to each rule bound, and multiple bounds with different severity can be defined for a rule. The severity information of failed rules is returned to the scheduler. Based on this information, the scheduler might, e.g., decide to interrupt an ETL process or to alert the DWH team.

Each rule's SQL statement can have *multiple parameters*, which are set at execution time. These parameters can for example be used to determine the range of data to be checked. In this way, a quality rule running after an ETL job might be limited to check only the new records in a fact table.

A. Sample rule

In the following, we show how a rule that checks NULL-values does look like. The target is to check the number of NULL value in the middle name in the customer records. As the typical percentage of people that have a middle name varies from country to country, we calculate the values per country. Thus, the country code is used as a dimension value and one result record per country is generated. The two base value fields are used to count the overall number of customers in each country and the number of customers without middle name, respectively. The result value is the percentage of customers in each country without middle name.

```
SELECT
  trunc(sysdate, 'dd') Result_date,
  countrycode dimValues,
  sum(1) baseValue1,
  sum(case when middlename is null
           then 1
           else 0 end) baseValue2,
  round(sum(case when middlename is null
                 then 1 else 0 end) /
        sum(1) * 100) resultValue
FROM customer
WHERE created >
      to_date($date$, 'YYYY-MM-DD')
GROUP BY countrycode
```

Fig. 2. Sample rule code

The SQL for this check is shown in Fig. 2. The result value is then checked against different bounds that are defined on a per-country basis. The rule uses a parameter `$date$` that is used to narrow the check to customers which have been created after the specified date.

```
SELECT
  trunc(sysdate, 'dd') Result_date,
  §reftable1_refdimension1§ dimValues,
  sum(1) baseValue1,
  sum(case when §reftable1_refattribute1§
               is null then 1
          else 0 end) baseValue2,
  round(
   sum(case when §reftable1_refattribute1§
                is null then 1
           else 0 end) /
   sum(1) * 100) resultValue
FROM §reftable1§
WHERE §reftable1_refattribute2§ >
      to_date($date$, 'YYYY-MM-DD')
GROUP BY §reftable1_refdimension1§
```

Fig. 3. Sample template code

B. Templating

In typical environments, there is often a need to define a number of equivalent rules over a large number of tables and attributes. To accommodate for this requirement, we implemented a *template concept*.

A template looks quite similar to a normal rule. It contains an SQL statement producing the same set of standard columns, and it might also contain bound definitions. However, instead of the target table and attribute names, the template's SQL statement contains special markers. For attributes, these markers declare the purpose of the attribute within the rule. Once the user has defined a template, she can instantiate it for multiple sets of tables and attributes. During this process, she either defines new bounds or uses the predefined bounds from the template for the generated rules. The engine forwards rule parameters defined within the template to the generated rules.

C. Example continued

The *sample* statement is a good candidate for a template. In the template, there is another type of parameters called template parameters that are replaced at template instantiation (i.e., rule creation time). These are used to define placeholders for the table and attribute names, like §reftable1§ (cf. Fig. 3).

A *GUI* assists unexperienced users with defining the template parameters, as shown in Fig. 1. In this dialog, the GUI reads the database catalog and lets the user map the template parameters to catalog objects. E.g., §reftable1§ is replaced with sales_fact. Note though that in the current version of the system data type integrity between catalog objects and template parameters is not automatically enforced by the system. I. e. the expert assigning catalog objects to template parameters has to take care that the data types are compatible. If they are not, the execution of the rule will fail with a corresponding SQL error message that will be recorded as a rule execution result in the result repository (cf. Section V). This issue never caused problems in the actual implementations of the system so far, but is a candidate for a future extension, e. g. by providing the opportunity for the template developer to place hints on the expected data type in the template that can assist the person actually creating the rules in the selection process.

IV. Rule Examples

In this section, we illustrate how the rule definition language introduced in the previous section can be used for advanced quality checks. In order to do so, we give examples for quality rules and show how they can be defined within the Data Checking Engine. The section has two purposes: On the one hand, we want to demonstrate that the DCE supports well-known rule types from the data quality literature. On the other hand, we want to introduce new rule types whose capabilities to detect possible quality problems are more sophisticated compared with the well-known rules. We use the AdventureWorks2008DW (in short AWDW) database from Microsoft [28] for our examples.

Data rules are found either by profiling and mining existing data, or by looking at business rules. The mining approach assumes that you either have a data set that is a-priori known to be correct, or you have to do outlier detection and data cleansing on the way. Currently, mining is out of scope for the DCE. This means that we performed the rule mining using external tools (in our case the GNU R tool). However, we plan to integrate this step further with the DCE system in the future, cf. Section VIII.

In general, we distinguish two kinds of rules. Hard rules are those that can clearly identify wrong data. As an example, the violation of a pattern for product codes confirms that the given product code is apparently incorrect. In the same sense, testing a foreign key relationship is a hard constraint. However, there are many quality problems that cannot be detected using hard rules. For this, we need another kind of rule, which we call *value rules*, according to Jack Olsen [5]:

There are additional tests you can construct that point to the presence of inaccurate data that are not as precise in establishing a clear boundary between right and wrong. These are called value rules.

We are going to present some examples of this kind of rules in the last subsections. However, we start with examples for simpler rules that are hard constraints.

A. Pattern for customer alternate key

```
SELECT
  trunc(sysdate, 'dd') Result_date,
  §reftable1_refdimension1§ dimValues,
  0 baseValue1,
  0 baseValue2,
  CASE WHEN
    regexp_like(§reftable1_refattribute1§,
                '$regexp$')
  THEN 1 ELSE 0 END resultValue
FROM §reftable1§
```

Fig. 4. Template for regular expression checking

In AWDW, each customer has an attribute CUSTOMERALTERNATEKEY that contains the business

key consisting of the letters 'A' and 'W' followed by 8 digits. Such a pattern can be validated using a regular expression.

The template shown in Fig. 4 is used for all rules that check regular expressions. To check the alternate key, we instantiate this template using the `dim_customer` table and set the `$regexp$`-parameter to `^AW[0-9]{8}$`.

B. Consistency of translated attributes

```
SELECT
  trunc(sysdate, 'dd') Result_date,
  t1.§reftable1_refdimension1§ dimValues,
  0 baseValue1,
  0 baseValue2,
  count(t2.§reftable1_refdimension1§)
    resultValue
FROM §reftable1§ t1
LEFT JOIN §reftable1§ t2
  ON t1.§reftable1_refattribute1§ =
     t2.§reftable1_refattribute1§
 AND t1.§reftable1_refattribute2§ !=
     t2.§reftable1_refattribute2§
GROUP BY t1.§reftable1_refdimension1§
```

Fig. 5. Template for functional dependency checking

In the AWDW dimension tables, some descriptions are present in multiple languages. As the same English text is expected to have always the same translation, a functional dependency (FD) is present. To check FDs, we have written a template (see Fig. 5). The template searches for tuple combinations that agree upon the first attribute's value and disagree upon the second attribute. For each tuple, the number of tuples that have a different value in the second attribute is counted.

We instantiated the template, e.g., for the attributes `ENGLISHEDUCATION` and `SPANISHEDUCATION` in the `dim_customer`-table. For each tuple in the table, the rule counts the number of tuples that match the current tuple with respect to `ENGLISHEDUCATION` but disagree in `SPANISHEDUCATION`. We have modified the text for `SPANISHEDUCATION` in a single tuple. As there are 5099 tuples that share the same text in `ENGLISHEDUCATION`, each of these tuples gets a resultValue of 1, while the modified tuple gets a value of 5098. Although the probability is high that the translation is correct for 5099 tuples and it is wrong for only a single tuple, the rule cannot make this distinction. Thus, the bound for the rule is zero, meaning each of the tuples is regarded to be a potential quality error.

C. Sales count of clothing and accessories

Now we start with examples for value rules. First, we are going to check whether the sales count (items per day) of the two product categories clothing and accessories is reasonable. A scatter plot of the counts per day in Fig. 6 indicates that there is a correlation between the two variables. Indeed, the correlation coefficient is 0.7469.

We are going to exploit this to build a quality rule. The idea is to estimate a bivariate normal distribution from the data

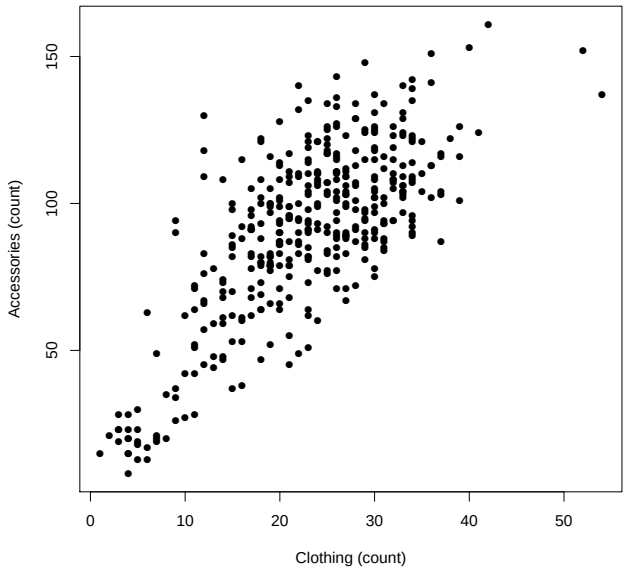


Fig. 6. Item sales per day in two categories

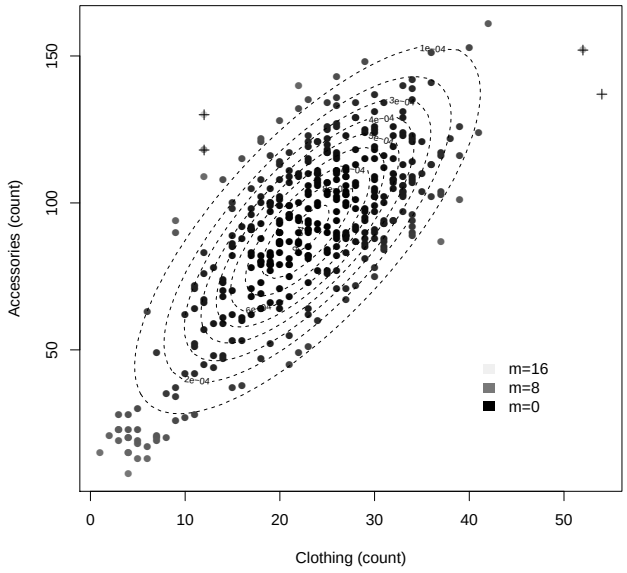


Fig. 7. Estimated distribution and distance

and to use the squared Mahalanobis distance from the center of the distribution as an outlier score for each data point. The estimation uses the variance-covariance matrix S of the data as an estimate for the covariance and the sample mean of the data $\bar{x}$ as the mean of the distribution. The contour lines of the density of the resulting distribution are displayed in Fig. 7.

The squared Mahalanobis distance is calculated using the following equation (see [29, p. 662]). Note that we assume

that x is a column vector.

$$m^2 = (x - \bar{x})^T S^{-1} (x - \bar{x}) \quad (1)$$

The distance values are shown in the figure using a grey scale. An important property of the Mahalanobis distance is that it takes the correlation between the attributes into account. As an example, the point $(40, 153)$ is not an outlier, while the point $(12, 118)$ is an outlier although it is more close to the center in terms of a Euclidean distance. This is because $(40, 153)$ better fits with the distribution of the data.

```
SELECT trunc(sysdate, 'dd') Result_date,
       key dimValues,
       x1 baseValue1,
       x2 baseValue2,
       msq resultValue
FROM
(SELECT (xn1*(xn1*$si11$ + xn2*$si21$) +
         xn2*(xn1*$si12$ + xn2*$si22$))
        msq,
        x1, x2, key
 FROM
 (SELECT x1-$xm1$ as xn1,
         x2-$xm2$ as xn2, x1, x2, key
  FROM
  (SELECT §reftable1_refattribute1§ x1,
          §reftable1_refattribute2§ x2,
          §reftable1_refdimension1§ key
   FROM §reftable1§
  )
 )
)
```

Fig. 8. Template to calculate Mahalanobis distances

The rule template in Fig. 8 calculates the squared Mahalanobis distance. It is based on a view `c34` that aggregates the sales count in both categories on a daily basis. The entries of S^{-1} are provided as parameters `$si11$`, etc. and $\bar{x}$ is provided using the parameters `$xm1$` and `$xm2$`.

We instantiated the template and used a bound of 9 on the distance. So each data point with a distance greater than 9 is declared to be suspicious. The corresponding points which are detected as errors by the rule are marked with a plus in Fig. 7. Please note that these errors have to be interpreted differently compared to the previous errors. Errors reported by value rules are unusual values that have to be investigated further by domain experts but do not immediately mean that a data quality issue has been detected.

D. Age distribution of customers

As another example for a value rule, we want to check whether the ages of our customers are reasonable. We assume that the current age distribution is correct and use it as a reference distribution. Our goal is to develop a quality rule that generates a warning when the distribution changes significantly. To achieve this, we start by creating a view that displays the current age distribution. The view is called `customer_age`. The distribution initially looks as shown in Fig. 9 (light gray bars labeled "original"). Note that in fact the distribution shows that AdventureWork's customers

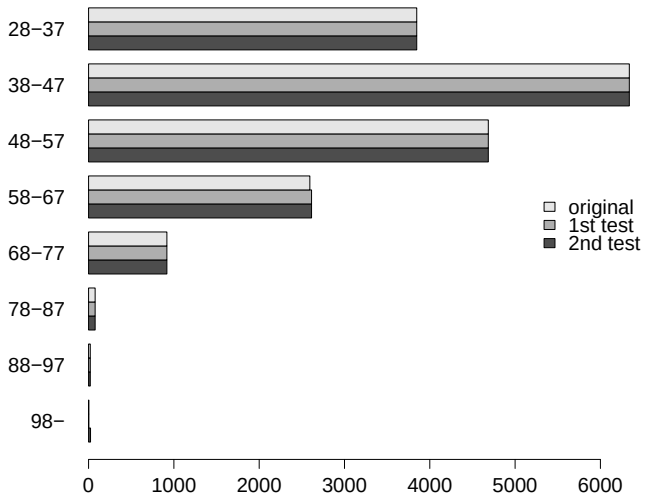


Fig. 9. Original age distribution and test modifications

```
SELECT trunc(sysdate, 'dd') result_date,
       1 dimValues,
       0 baseValue1,
       0 baseValue2,
       sum((N1-E1)*(N1-E1)/E1 +
           (N2-E2)*(N2-E2)/E2) resultValue
FROM (
SELECT nvl(N1, 0) N1, nvl(N2, 0) N2,
       N1SUM * (nvl(N1, 0) + nvl(N2, 0))
               / (N1SUM+N2SUM) E1,
       N2SUM * (nvl(N1, 0) + nvl(N2, 0))
               / (N1SUM+N2SUM) E2
FROM
(SELECT §reftable1_refattribute1§ key,
        §reftable1_refattribute2§ N1
 FROM §reftable1§) D1
FULL OUTER JOIN
(SELECT §reftable2_refattribute1§ key,
        §reftable2_refattribute2§ N2
 FROM §reftable2§) D2
ON (D1.key = D2.key),
(SELECT sum(§reftable1_refattribute2§)
        N1SUM FROM §reftable1§) N1,
(SELECT sum(§reftable2_refattribute2§)
        N2SUM FROM §reftable2§) N2
)
```

Fig. 10. Template for χ^2 homogeneity test

are quite old, which might already indicate a data quality problem. However, we ignore this for the sake of the example and assume that the initial distribution is correct.

We now store a snapshot of the view in the table `customer_age_ref`. This table will serve as a reference for the current age distribution. Now we can use a χ^2 homogeneity test to see whether the current customer's ages are similarly

distributed. The test statistic is defined as follows:

$$\chi^2 = \sum_{j=1}^{k}\sum_{i=1}^{m}\frac{(n_{ij} - E_{ij})^2}{E_{ij}} \quad (2)$$

In this formula, k is the number of samples to be compared. As we have two samples (the reference age distribution vs. the current distribution), we have $k = 2$. Furthermore, m is the number of groups, in our case $m = 8$ as we have eight age groups. n_{ij} is the number of customers in sample j belonging to age group i. E_{ij} is the expected number of customers in sample j in age group i. It is calculated using the margins:

$$E_{ij} = \frac{n_{i\cdot}n_{\cdot j}}{n} \quad (3)$$

Here, $n_{i\cdot}$ is the overall number of items in group i in both samples and $n_{\cdot j}$ is the size of sample j.

Again, we define a rule template to calculate the statistic. It is shown in Fig. 10.

The result of this statement is the test statistic for the χ^2 test. Its value has to be compared with an appropriate quantile of the χ^2 distribution: $\chi^2_{(k-1)(m-1);0.95} = \chi^2_{7;0.95} = 14.067$. We use this value as an upper bound. All values above this one will generate a quality error.

When we initially run the rule, the output is (as expected) 0. This is due to the fact that we compare two identical samples. To test the rule, we first insert 20 customers of age 59. They are in age group 58-67, which already has nearly 2600 customers. The rule now yields a value of 0.066, which is way below our bound, as expected. Now we start to insert more unusual customers of age 99. As we insert 5 of them, the value already becomes 2.33. With 15 more customers of this age, we reach 15.418, which generates a quality error.

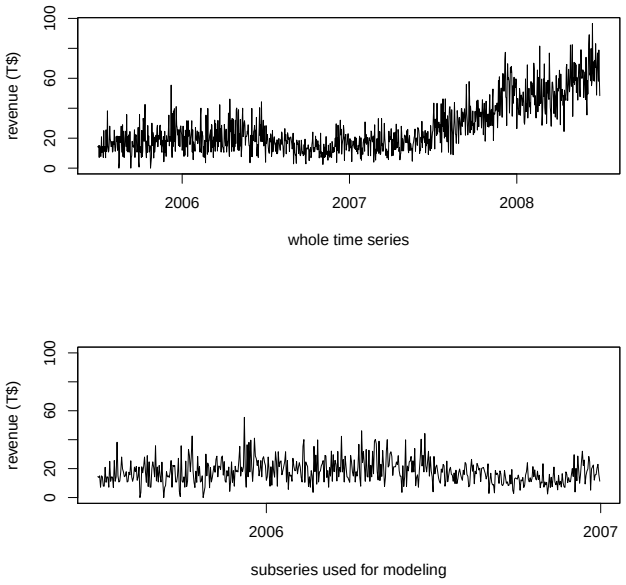


Fig. 11. Daily internet revenue

E. Check the revenue time series

In our final example, we check the overall daily Internet revenue of the AdventureWorks company. It can be calculated by aggregating the facts in `fact_internet_sales`. Again, the underlying idea of the rule is to define a model for the data and to check that the new data does not contradict the model significantly. For this example, we use the data up to the end of 2006 to define the model and then check the data from 2007 on against the model. Fig. 11 shows the complete time series and an enlarged plot of the part used for modelling.

In this case, a time series model is appropriate and we use the ARIMA family of models [30]. These are stochastic models that are composed of an auto-regessive part (AR), an integration part (I), and a moving average part (MA). When the integration part is present, the differenced time series is used instead of the original one. The basis for the series is a series of random components. They are assumed to be independent and normally distributed. The AR part captures dependencies between the current value and the preceding values, while the MA part is used to model a moving average of the past random components.

A first look at the plots indicates that the series is non-stationary, as there seems to be a trend, at least from mid-2007 on. The variance looks fairly large and there is no sign of seasonality.

We use the `auto.arima` function of R to select an appropriate ARIMA model and to estimate the parameters. The result is an ARIMA(0,1,2) model. This means that no AR part is present, the original series is differentiated once, and the MA part averages over the past two random components. The following coefficients are estimated:

$$y_t - y_{t-1} = e_t + -1.0012e_{t-1} + 0.0626e_{t-2} \quad (4)$$

The random components are denoted by e_t. We can see that the negative last random value has a huge influence on the current observation (coefficient -1.0012). The estimated standard deviation is $\hat{s} = 7951.597$. The AIC value is rather large (11407).

The basic idea of the rule is to use the model to calculate each day a one step forecast and to check whether the newly observed value is within the forecast interval. If it is outside, then we generate an error, as the value is rather unlikely. For the forecast, we use equation (4) and set $e_t = 0$, as we know nothing about the current random component. The past two values e_{t-1} and e_{t-2} are estimated using the past residuals (i.e., the differences between the past forecasts and observed values). We denote the residuals with $\hat{e}_{t-1}$ and $\hat{e}_{t-2}$. This yields the following equation for the point forecast $\hat{y}_t$:

$$\hat{y}_t = y_{t-1} + -1.0012\hat{e}_{t-1} + 0.0626\hat{e}_{t-2} \quad (5)$$

The 95% forecast interval can be computed as $\hat{y}_t \pm \hat{s} * 1.96$.

In order to build a DCE rule using this model we have to store the past residuals so that we can access them during the current rule execution. We use the DCE repository's rule outcome table for this task. In this way, each day's run stores the current residual in the repository. We have to bootstrap the rule by feeding at least two residuals in the results table so that the rule can start to calculate the next forecasts. These

past residuals are included in the R object generated by the `auto.arima` function. Currently, we have to store them in the repository by hand. Later, we plan to automate this task in a profiling phase.

```
SELECT
  trunc(sysdate, 'dd') Result_date,
  datekey dimValues,
  extendedamount baseValue1,
  0 baseValue2,
  get_residual('$checkdate$') resultValue
FROM daily_revenue
WHERE fulldatealternatekey =
      to_date('$checkdate$', 'YYYY-MM-DD')
```

Fig. 12. Rule to check a time series

We implemented the forecast equation (5) as a stored function `get_residual` within the database. The function only returns the residual, as this is all we need to check the rule bounds and to prepare for the next run of the rule. Using this stored function, the rule's SQL is quite simple, as shown in Fig. 12. The view `daily_revenue` calculates daily aggregates of the internet sales revenue.

The rule uses a parameter `$checkdate$` that specifies the target date. Each rule run checks a single day. As the rule depends on the past two runs we have to ensure that the rule runs every day without omissions.

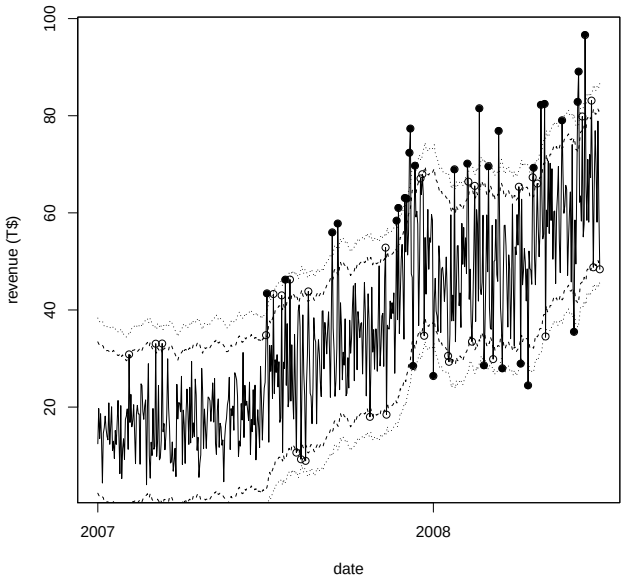


Fig. 13. Prediction intervals and warned values

Fig. 13 shows the second part of the time series and the forecast intervals calculated from the above formula for 95% (dashed) and 99% (dotted). Circles indicate data points that are outside the 95%; filled circles indicate those that are also outside the 99% interval. Depending on the bounds specified in the rule, the DCE reports these values.

V. ARCHITECTURE

In this section, we will illustrate the system architecture of the current DCE implementation, which is capable of checking the previously explained data quality rules. We also show how the DCE fits into a typical enterprise DWH implementation. Fig. 14 shows an overview of the DCE overall architecture. The DCE itself is organized as a classical three-tier application. It interacts with the enterprise data warehouse system in order to compute quality indicators. Also, results of the data quality checks may be propagated into another external database system, the data quality data mart. This database in itself is also organized as a data mart and provides long term storage of computed data quality indicators in order to be used for long term analysis of enterprise wide data quality. In a sense it is a meta-data warehouse for data quality. There is also an external scheduling component (typically standard system scheduling capabilities), which triggers computation of data quality indicators at previously defined points in time.

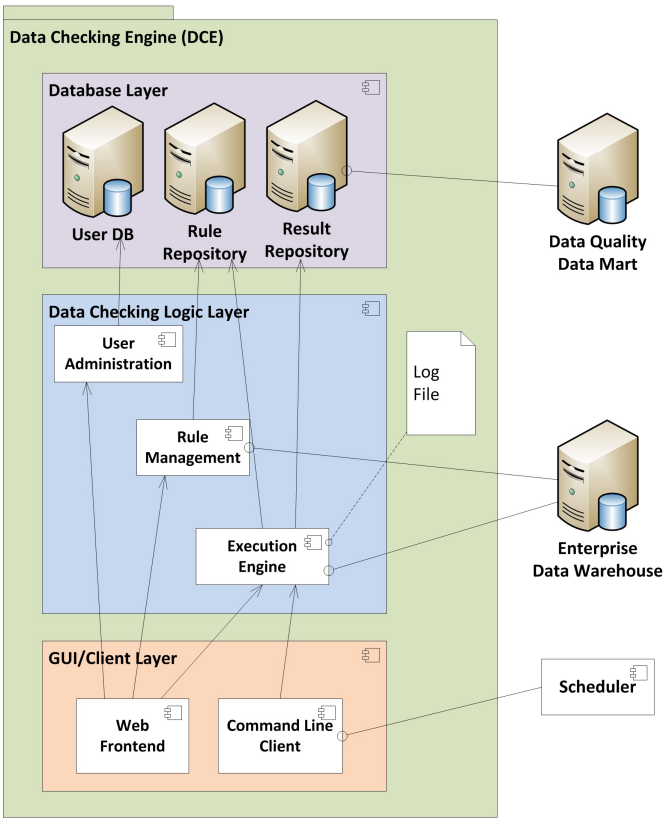


Fig. 14. Data checking engine architecture overview

Within the DCE itself the main entry point for data quality managers is the GUI of the DCE web application (shown at the bottom of Fig. 14). The GUI is used to manage users of the DCE application, to manage data quality rules, and to manage data rule executions. As typically the execution of data quality checks is not triggered manually, there is also a command-line client library for the rule execution engine that is triggered by an external scheduler. The schedule to be used is managed in the web application as well.

The main data checking business logic can be found in the middle tier. This logic is used by the web application

as described above. Note that there is a strict separation between user management, rule management and rule execution management in the middle tier as well. Whereas the user administration component provides standard functionality, note that the rule management component contains advanced features. For instance the template mechanism described in the previous section is implemented here.

The execution engine is also managed by the web application: on the one hand, rules can be manually executed from the web application, on the other hand, scheduled execution can be defined here.

During rule execution, the engine replaces the parameters in the rule's SQL statement with their current values and then runs the statement using the target database. Thus, moving large amounts of data into the DCE engine is avoided. The result of the SQL statement is then further processed. This includes checking the currently applicable bounds and testing their severity.

In the execution engine, it is also defined, which rules are executed on what data warehouse database under whose privileges. Note that multiple different data warehouses (or database systems) may be used as source, because the connection information is also managed by the web application.

Note that the performance of the engine itself is not critical. All rules are finally translated to SQL and executed on the target database. Compared to the execution time of the SQL statement (which perhaps runs on large data sets), the preparation phase and the interpretation of the results, which includes bound checking, is negligible. Thus, the scalability of the DCE depends heavily on the scalability of the target database. In case of performance problems, typical database tuning options like indexing or materialization of views are used.

Finally, the database layer consists of three separate areas:

- Rule repository, which holds the data quality rules as well as base templates
- Result repository holding results of rule execution
- User database which is used for access management to only the DCE itself

Once results of the executed data quality rules have been stored in the result repository they may be propagated to the data quality data mart that aggregates the results into quality indicators.

This data mart is not part of the DCE but located within the standard DWH infrastructure of the company. Thus, standard interfaces such as reporting and BI tools can be used to further present and analyze the data quality status. This way the additional effort for data quality monitoring can be kept minimal as access to data quality indicators follows well established processes and uses well-known tools, which are used for regular monitoring of enterprise performance indicators as well. In addition, the concept of viewing data quality indicators similar to regular performance indicators is very fitting, as these have to be tracked accordingly in order to ensure reliability of data in the data warehouse. Ultimately, this is necessary to make the right entrepreneurial decisions based on reliable information.

VI. Data Quality Metrics

The result repository contains large amounts of specific results that individually describe only a very small fraction of the overall data quality of the DWH. In order to get a quick overview of the quality level, a small set of metrics that aggregate the rule results is required.

In the literature, there are various approaches to define data quality indicators, for example [31]. Thus, we decided to provide a flexible approach that enables the user to define her own indicator hierarchies. The engine stores indicator definition meta data and calculates the resulting indicator values.

An important issue here is to take incremental checks into account. As an example, consider a rule that checks the number of dimension foreign keys in a fact table that reference a dummy instead of a real dimension entry. As the fact table is large, the daily rule just checks the new fact records loaded in the previous ETL run. Thus, the indicator has to aggregate over the current and past runs to provide an overall view of the completeness of the dimension values.

VII. DCE 'Look and Feel' Impressions

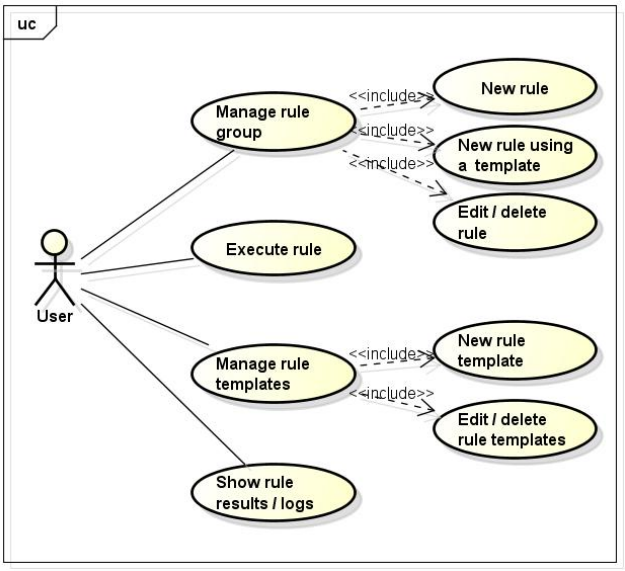


Fig. 15. End user – Major use cases

Based on the above discussed concepts and architecture we implemented the overall DCE system. To give an impression how DCE actually looks like from a usage perspective, we will illustrate in this section some selected use cases and screen shots from the DCE GUI.

A. Use Cases

We can distinguish between two main types of users within DCE, namely end users and administrators.

End users are the actual rule authors. Based on their domain knowledge, they formulate DCE rules as described above. Respectively, their main use cases (after their login to DCE) are 'show rule group', 'input rules' maybe based on an existing

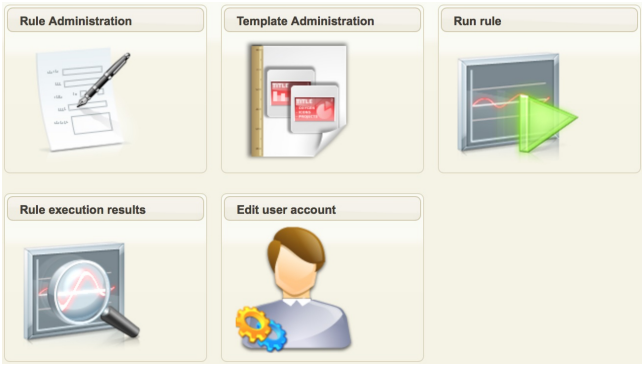


Fig. 16. User's central menu

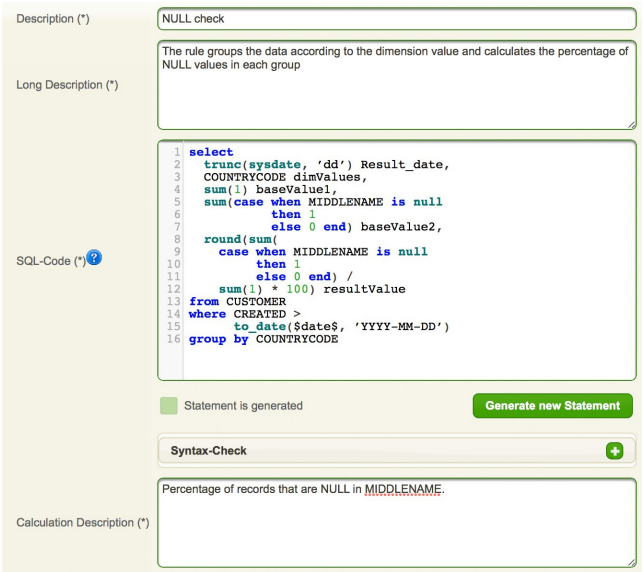


Fig. 17. User's rule insert / edit screen

one, 'edit rule', and 'show rule results'. As explained, rules could be based on rule templates. The main use cases are illustrated in Fig. 15.

Administrators have as their main responsibilities the management of user accounts, database connections, and general settings.

Name		Start Date	End Date	Parameters	Value	Comparison	Result
NULL check (Group ID: 301)		2014-02-27 09:12:13.952	2014-02-27 09:12:14.078				Passed
NULL check (Group ID: 301)		2014-02-27 09:14:57.832	2014-02-27 09:14:57.953				Passed
NULL check (Rule ID: 401)		2014-02-27 09:14:57.87	2014-02-27 09:14:57.943	date: 2010-01-01			Passed
AU					42.0	<= 50.0; 0.0	Passed (0)
CA					42.0	<= 50.0; 0.0	Passed (0)
DE					42.0	<= 70.0; 0.0	Passed (0)
FR					42.0	<= 70.0; 0.0	Passed (0)
GB					43.0	<= 50.0; 0.0	Passed (0)
US					42.0	<= 50.0; 0.0	Passed (0)

Fig. 18. Execution results 1

Name	Description	Start Date	End Date		Comparison	Result
agetest (Group ID: 153)		2014-01-29 18:37:56.572	2014-01-29 18:37:56.835			Passed
agetest (Group ID: 153)		2014-01-29 19:36:50.199	2014-01-29 19:36:50.507			Passed
agetest (Group ID: 153)		2014-01-29 19:37:15.881	2014-01-29 19:37:16.561			Passed
homogeneity test (Rule ID: 205)	uses chi squared	2014-01-29 19:37:16.298	2014-01-29 19:37:16.556			Passed
1				0.065	< 14.06714; 0.0	Passed (0)
agetest (Group ID: 153)		2014-01-29 19:37:52.25	2014-01-29 19:37:52.982			Passed
homogeneity test (Rule ID: 205)	uses chi squared	2014-01-29 19:37:52.704	2014-01-29 19:37:52.976			Passed
1				2.332	< 14.06714; 0.0	Passed (0)
agetest (Group ID: 153)		2014-01-29 19:38:55.73	2014-01-29 19:38:56.057			Error
homogeneity test (Rule ID: 205)	uses chi squared	2014-01-29 19:38:55.778	2014-01-29 19:38:56.052			Error (0)
1				15.41	< 14.06714; 0.0	Error (0)

Fig. 19. Execution results 2

B. Screen Shots

To give an impression of the 'look and feel' of the DCE GUI, we show a few screen shots here.

Fig. 16 shows the central menu options for a DCE end user, namely rule and template management, rule execution, execution results inspection, and edit user account.

A typical DCE end user task is the insertion of new rules. Fig. 17 shows the corresponding screen, which allows a DCE end user to edit new rules.

Two typical result screens from rule execution are shown in Fig. 18 and Fig. 19. In Fig. 19 results from failed rule checks (errors) are shown as well.

In addition, further options exist, e. g., to edit rule templates, to group rules, and options for administrators as well.

VIII. Conclusion and Future Work

In this paper, we have presented a motivation why complex data quality rules are required in most practical enterprise application scenarios. We have developed a rule definition language with a two-stage templating mechanism in order to be able to express most of these rules for SQL-based data sources. The language along with the templating mechanism is both powerful enough to express complex rules as well as simple enough to be used by domain experts in practice. This has been proven by a prototypical implementation based on the described architecture. This tool has been validated by interviewing teams of different DWH projects and building project specific prototypical setups in a real world environment. Our engine has been able to support their quality monitoring requirements. Especially, the flexibility in rule definition was appreciated. We have not only detected quality problems on tuple level but also more complex issues, e.g., checking the trend of indicators stored in a fact table. As expected, our template mechanism has proven to be an important way to simplify rule definition.

The engine keeps a comprehensive history of rule results *and* rule meta data, which allows to monitor data quality over time and to check whether quality improvement projects were successful. This quality data is exposed to external BI tools for

reporting and further analysis. This integration of quality data into tools for further analysis will be significantly improved and simplified in the future. We aim to provide semi-automated aggregation features that provide the data quality manager in an enterprise with a set of traffic lights on a dashboard-like portal in order to get an impression of the overall data quality quickly and easily.

An important consequence of the flexibility of our approach is that the DCE can also be used during DWH/ETL development to test the result processes. The testing rules developed during this project phase may also be used during normal operation later on, reducing the overall cost of data quality. Practical experiences with this kind of dual use of rules will be gathered in the future. An advantage of this approach might be an improved quality of the quality rules themselves as their definition will be based on thoroughly developed concepts.

Our approach is currently working on any relational database system. In the future, we plan to also integrate data in other data sources such as Big Data systems like Hadoop and other NoSQL database systems, as more and more relevant data will be stored there. Thus, data quality should be monitored there as well. As there is currently no universal query language standard like SQL in the relational sector, we will have to devise a flexible way to cope with various rule definition languages and/or define a generic rule definition language together with automated translations into the languages of the source systems.

We will also work further on the process of determining quality rules. For newly defined rules the data quality manager faces the issue that it is difficult to determine whether there is an actual data quality problem or an inaccuracy in the rule. Consequently, we plan to further explore the possibility to semi-automatically derive data quality rules by advanced data profiling methods. These may even be enhanced by integrating data mining processes.

Finally, there is still a need for even more advanced data quality rules that are based on more sophisticated statistical models (both static and dynamic) as well as multi-variate time series data. While basic ideas for this have been presented in Section IV, in the future we would like to extend this approach and also reduce the manual effort by semi-automated steps.

REFERENCES

[1] F. Heine, C. Kleiner, A. Koschel, and J. Westermayer, "The data checking engine: Monitoring data quality," in *DATA ANALYTICS 2013, The Second International Conference on Data Analytics*, 2013, pp. 7–10.

[2] S. Sadiq, Ed., *Handbook of Data Quality*, 1st ed. Springer, 2013.

[3] W. Fan and F. Geerts, *Foundations of Data Quality Management*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[4] C. Batini and M. Scannapieco, *Data Quality: Concepts, Methodologies and Techniques*, 1st ed. Springer, 2006.

[5] J. Olson, *Data Quality. The Accuracy Dimension*. Morgan Kaufmann, 2002.

[6] R. Kimball and J. Caserta, *The data warehouse ETL toolkit*. Wiley, 2004.

[7] L. Bertossi and L. Bravo, *Handbook of Data Quality*, 1st ed. Springer, 2013, ch. Generic and Declarative Approaches to Data Quality Management, pp. 181–211.

[8] P. Z. Yeh and C. A. Puri, "An efficient and robust approach for discovering data quality rules," in *22nd International Conference on Tools with Artificial Intelligence*, 2010.

[9] F. Chiang and R. J. Miller, "Discovering data quality rules," in *Proceedings of the VLDB 08*, 2008.

[10] I. P. Fellegi and D. Holt, "A systematic approach to automatic edit and imputation," *Journal of the American Statistical Association*, vol. 71, no. 353, pp. 17–35, 1976.

[11] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, "Towards certain fixes with editing rules and master data," *The VLDB Journal*, vol. 21, no. 2, pp. 213–238, Apr. 2012. [Online]. Available: http://dx.doi.org/10.1007/s00778-011-0253-7

[12] S. Song and L. Chen, "Differential dependencies: Reasoning and discovery," *ACM Transactions on Database Systems*, vol. 36, no. 3, pp. 16:1–16:41, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2000824.2000826

[13] S. Brüggemann, "Addressing internal consistency with multidimensional conditional functional dependencies," in *International Conference on Management of Data COMAD 2010, Nagpur, India*, 2010.

[14] F. Korn, S. Muthukrishnan, and Y. Zhu, "Checks and balances: Monitoring data quality problems in network traffic databases," in *Proceedings of the 29th VLDB Conference, Berlin*, 2003.

[15] M. L. Lee, T. W. Ling, and W. L. Low, "Intelliclean: A knowledge-based intelligent data cleaner," in *ACM SIGKDD, Boston, 2000*, 2000.

[16] W. Fan, H. Gao, X. Jia, J. Li, and S. Ma, "Dynamic constraints for record matching," *The VLDB Journal*, vol. 20, pp. 495–520, 2011.

[17] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, 2007.

[18] K. Sattler, S. Conrad, and G. Saake, "Adding conflict resolution features to a query language for database federations," in *Proc. 3nd Int. Workshop on Engineering Federated Information Systems, EFIS'00, Dublin, Ireland, June*, 2000, pp. 41–52.

[19] J. Barateiro and H. Galhardas, "A survey of data quality tools," *Datenbank-Spektrum*, vol. 14, 2005.

[20] F. Naumann, "Data profiling revisited," *SIGMOD Record*, 2013.

[21] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882

[22] C. C. Aggarwal, *Outlier Analysis*. Springer, 2013.

[23] K. Yamanishi and J. Takeuchi, "A unifying framework for detecting outliers and change points from non-stationary time series data," in *Proceedings of the Eighth ACM SIGKDD-02*, 2002.

[24] C. S. Hilas, I. T. Rekanos, S. K. Goudos, P. A. Mastorocostas, and J. N. Sahalos, "Level change detection in time series using higher order statistics," in *16th International Conference on Digital Signal Processing*, 2009.

[25] S. Sarawagi, R. Agrawal, and N. Megiddo, "Discovery-driven exploration of olap data cubes," in *Advances in Database Technology — EDBT'98*, 1998.

[26] E. Müller, M. Schiffer, and T. Seidl, "Statistical selection of relevant subspace projections for outlier ranking," in *27th IEEE International Conference on Data Engineering (ICDE)*, 2011.

[27] C. Ordonez and Z. Chen, "Evaluating statistical tests on olap cubes to compare degree of disease," *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, no. 5, 2009.

[28] Microsoft. (2014, Feb) Microsoft sql server database product samples. [Online]. Available: http://msftdbprodsamples.codeplex.com

[29] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Pearson, 2006.

[30] A. C. Harvey, *Time Series Models*. Pearson Education, 1993.

[31] B. Heinrich, M. Kaiser, and M. Klier, "Metrics for measuring data quality - foundations for an economic oriented management of data quality," in *Proceedings of the 2nd International Conference on Software and Data Technologies (ICSOFT). INSTICC/Polytechnic Institute of Setúbal*, J. Filipe, B. Shishkov, and M. Helfert, Eds., 2007.

Conceptual Modelling in UML and OWL-2

Jesper Zedlitz and Norbert Luttenberger
Christian-Albrechts-Universität
Kiel, Germany
Email: {j.zedlitz, n.luttenberger}@email.uni-kiel.de

***Abstract*—Both OWL-2 and UML static class diagrams lend themselves very well for conceptual modelling of complex information systems. Both languages have their advantages. In order to benefit from the advantages and software tools of both languages, it is usually necessary to repeat the modelling process for each language. We have investigated whether and how conceptual models written in one language can be automatically transformed into models written in the other language. For this purpose we investigated differences and similarities of various model elements (such as element type, data types, relationship types) in static UML data models and OWL-2 ontologies. We provide a transformation for similar elements.**

***Keywords*—UML; OWL; conceptual modelling; model transformation; meta modelling**

I. Introduction

The Web Ontology Language (OWL) is mostly considered as a language for knowledge representation. However, it can also be used as a language for conceptual modelling of complex information systems. It can be used as a language to represent the entities of a certain domain, to express the meaning of various, usually ambiguous terms and to identify the relationships between them.

In this respect, OWL can be seen as a direct "competitor" to static class diagrams from Unified Modeling Language (UML). This kind of diagrams is often used for conceptual modelling, for example in the ISO 191xx series of standards. Both languages have their benefits. UML's visual syntax is easy to understand and there is a variety of software tools to choose from. OWL is backed up by formal logic and logical conclusions can be drawn on models using inference software (a.k.a. *reasoner*). In order to be benefited from the advantages and software tools of both languages, it is usually necessary to repeat the modelling process for each language. We have investigated whether and how conceptual models written in one language can be automatically transformed into models written in the other language.

In contrast to existing approaches, our transformation abstracts from the concrete syntax—in most cases a serialisation based on the Extensible Markup Language (XML)—and operates on the level of UML's and OWL's meta-models. This allows to show which model elements can be transformed and which cannot—independent of individual examples.

This article is the extended version of the paper published at SEMAPRO 2013 [1]. It is organized as follows. We start with a general overview of our approach. The following sections deal with certain kind of constituents: element types (Section III), data types (Section IV), relationship types (Section V), and constraints (Section VI). Each section describes how the particular kinds of model element is represented in UML and OWL and how it can be transformed from one language into the other. Section VII deals with the question of whether the transformation rules presented in the previous sections are correct and—within their previously specified limits—complete. Section VIII gives an overview of existing approaches for transformations between UML and OWL. In Section IX, we summarize our work and point out fields of future work.

II. Our Approach

In this section, we present the main ideas of a transformation of conceptual models on the meta model level by using a standardized declarative model transformation language.

In order to transform between OWL and UML models, it is necessary to find a "common third". This common third might be a transfer format, i.e., a common syntax. It might also be a common meta model, which permits a syntax independent transformation of model elements of one language into corresponding model elements of the other language.

A fundamental difference between OWL-2 and its predecessors is the fact that OWL-2 has a MOF-compliant meta model. OWL-2 ontologies can be processed not only as serialized XML documents, but also as MOF-based models. Therefore, all tools that are available for model transformations in the context of MOF can also be used to process ontologies.

A *model-based* transformation only operates on a syntactical level. In contrast, a *model* transformation offers access to both the models as well as the meta models [2, p. 28]. The transformation rules refer to the meta models only—hence the term "processing on meta model level". When writing the transformation rules, it is not necessary to know which models are going to be transformed later. Every model that is compliant with the (input) meta model can be processed using these transformation rules.

As a transformation language for MOF-based models, the Object Management Group (OMG) introduced the "Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT)". It is particularly well suited for our purposes. The fact that the name of the language contains "MOF" makes the close connection obvious. QVT includes two different language versions. For our purposes, the declarative QVT Relations (QVT-R) is more suitable:

- A declarative notation leads to compact transformation rules that require less code duplication.
- The developed rules, their interaction with each other- as well as the connection between the source and

the target model, may subsequently be analysed more effectively.

- During the execution of a QVT-R transformation so called *trace classes* are generated. These are useful for later analysis.
- QVT-R has a graphical syntax allowing a clear and easy to understand presentation of the transformation rules.

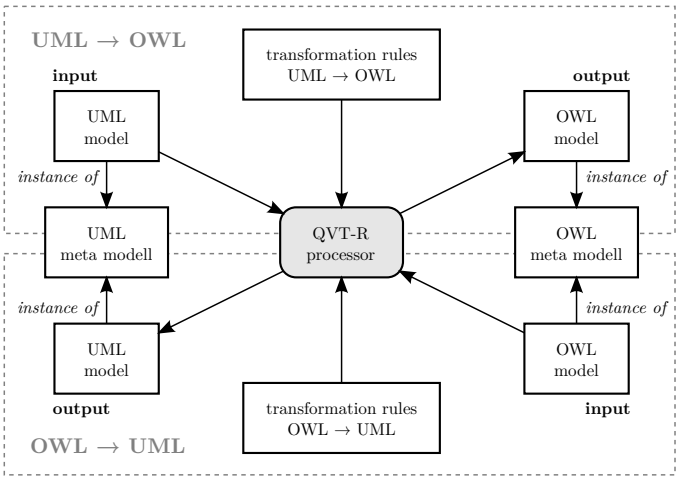


Fig. 1. Sketch of the transformation UML↔OWL. The upper part of the figure is read from left to right. The lower part is read from left to right.

Figure 1 sketches the complete transformation process for UML and OWL used in the context of this work. The upper half shows the UML → OWL transformation, the lower half the OWL → UML transformation. Common for both directions of transformation is the use of UML and OWL meta models. Before and after the actual model-to-model transformations, certain pre-transformations need to be done, especially when dealing with OWL ontologies. These pre-transformations map between concrete and abstract syntax. This can easily be done using the OWL API [3] or an XSLT script.

III. Element Types

Element types are among the most important components of a conceptual model: "Defining the entity types [...] is a crucial task in conceptual modeling" [4, p. 41]. Element types are also known as concepts and are eponymous for the term "conceptual model". Element types are used to group individual objects with the same or similar characteristics.

Both UML and OWL make an equal distinction between classes on the one side and "instances" resp. "individuals" on the other side. Because of this similarity, a transformation from UML classes into OWL classes is straight forward.

For the transformation from OWL to UML, one must distinguish between named—i.e, declared—classes on the one hand and unnamed class expressions (CE) on the other hand. A named class is the simplest case. In this case, a UML class is created with its name derived from the identifier of the class. Transformations of unnamed CE will be covered in later sections about generalization (ObjectUnionOf) and intersection (ObjectIntersectionOf).

A. Names

For an unambiguous identification, each element type must have a unique name within a model. The UML specification demands that every model element must have a unique name within its package. This ensures that each model element can be uniquely identified by specifying its name and the position in the package hierarchy. OWL uses Internationalized Resource Identifier (IRI) conforming to RFC 3987 to name element types as well as other elements of an ontology—instances of the OWL meta class `Entity`. All assigned names have global scope, regardless of the context in which they are used.

Since—unlike in UML—an OWL model element must have a unique name not only within a package but globally, such a globally unique name must be assigned during the transformation of any model element. Therefore, corresponding globally unique IRIs are generated during the transformation. For the transformation OWL → UML it is sensible to use the remaining characters of an IRI written in short form as the name of UML elements—assuming the prefix IRI is associated with the package. As a result the generated UML models will be easier to read. This is possible, because names in the UML must be unique only within a package.

B. Inheritance

A common situation in conceptual modelling is that the population of one element type is neccessarily also the population of a another element type. This is referred to as an inheritance relationship between these two element types.

Due to the very similar structure and semantics—especially the transitivity—of *Generalization* elements on the one hand and `SubClassOf` axioms on the other hand, a transformation from UML to OWL is easily possible. If both of the elements that are connected via an instance of the UML meta class *Generalization* can be transformed to OWL, the transformation of the *Generalization* instance is simple. An instance of the OWL meta class *SubClassOf* must be created during the transformation process. Since the newly created `SubClassOf` element is an axiom, it is necessary to connect it to the containing ontology.

For the transformation of an axiom of the form `SubClassOf(`C_c C_p`)` in the direction OWL → UML the following cases must be distinguished:

(a) Both sub-class C_c and super class C_p can be transformed into a UML class.

(b) At least one of the element types is a CE and the membership to that CE is described by necessary and sufficient conditions.

(c) Al least one of the element types cannot be transformed into a UML class due to other reasons (e.g. complementing).

In case (a) in which both element types can be mapped, a transformation is simple. Figure 2 shows an example for the transformation of a `SubClassOf` axiom in this simplest case. An instance of the UML meta class *Generalization* is used to create an inheritance relationship between the two transformed UML classes.

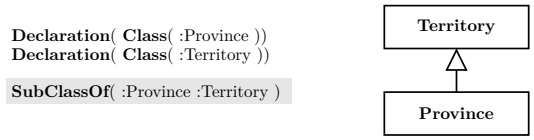


Fig. 2. Example for the transformation of a `SubClassOf` axiom.

If in case (b) one of the two element types is a CE defined by necessary and sufficient conditions, one has to distinguish whether it is the sub-class C_c or super-class C_p. If it is the sub-class C_c it is a case as shown in this example:

```
SubClassOf(
   DataMinCardinality( 1 :hasISBN )
   :Book
)
```

The element type C_c is defined by a formula ϕ. Every individual e is an instance of this element type if, and only if, it satisfies the formula:

$$C_c(e) \leftrightarrow \phi(e)$$

The fact that an individual e is an instance of C_p is denoted by $C_p(e)$. From the inheritance relationship

$$C_c(e) \rightarrow C_p(e)$$

immediately follow (by substituting $C_c(e)$)

$$\phi(e) \rightarrow C_p(e)$$

Meeting the formula $\phi(e)$ is therefore a sufficient condition that an object is an instance of the element type C_p. Since UML does not support automatic classification on the basis of sufficient conditions, this case is not transformable.

The situation in case (b) is different, if the element type defined by necessary and sufficient conditions appears as super-type C_p. Here is an example for this case:

```
SubClassOf(
   :Book
   DataMinCardinality( 1 :hasISBN )
)
```

Taken alone, the element type C_p could not be transformed into a UML class. However, the fact that the element type C_p occurs within in `SubClassOf` axiom as super-type makes the necessary condition of the CE a necessary condition of the sub element type C_c. This is shown in the following:

The element type C_p is defined by a formula ϕ. Every individual e is instance of this element type if, and only if, it satisfies the formula:

$$C_p(e) \leftrightarrow \phi(e)$$

The fact that an individual e is an instance of C_c is denoted by $C_c(e)$. From the inheritance relationship

$$C_c(e) \rightarrow C_p(e)$$

immediately follow (by substituting $C_p(e)$)

$$C_c(e) \rightarrow \phi(e)$$

To meet the formula $\phi(e)$, it is therefore a necessary condition that an object is an instance of the element type C_c. The example shown in the listing above can be read as: "Every book must have at least one ISBN." Such necessary conditions are in turn easily transformable into UML.

C. Abstract Elements

An element type is called abstract if it cannot be instantiated. Thus, one might assume, models containing abstract classes are by definition inconsistent. Wahler et al. provide a solution to this apparent problem in [5]. UML allows the definition of abstract classes that must not have direct instances. Instantiation is only allowed for subclasses. OWL knows no language construct to define that a class must not contain any individual directly, and only one of its subclasses is allowed to contain individuals.

Usually, abstract classes appear in connection to generalizations. In that context, it is possible to treat them as "normal" classes. However, the limitation remains that after the transformation from UML to OWL, it cannot be assured that an abstract class does not contain any direct instances.

During the transformation of instances of the OWL meta class `ObjectUnionOf` in connection with inheritance relationships, abstract element types play a role. This is discussed in the section about generalizations below.

D. Element types with fixed population

An element type with a fixed population consists of a predefined set of objects that make up the population of this element type. It is not possible to classify more objects as instances of the element type. UML has no way to specify that the population of an element type may only consist of a fixed set of objects. For data types, it is possible to define element types with fixed population by using an enumeration.

In OWL, elements with a fixed population can be defined for both individuals as well as for values of data types. For a fixed set of individuals, this is a CE. A predefined set of values is a data range. It is not necessary that the listed individuals are instances of the same element type. Similarly, the values listed do not have to belong to the same data type. Since, in UML one can only defined data types with fixed population, a transformation of the `ObjectOneOf`-CE is not possible. Both transformation directions for data types are described in the corresponding sections about enumerations.

E. Generalization

A generalization is an extension of the inheritance concept discussed above. Usually, more than two element types are put into relation. It is also possible to specify whether it is a complete and/or non-overlapping generalization.

In UML, the generalization of element types is represented similarly to the the inheritance of element types. Also, information about whether a generalization is complete and/or non-overlapping (disjoint) can be expressed in UML. Four

characteristics can be used for this: complete, incomplete, disjoint, or overlapping.

In addition to simple inheritance with the help of a `SubClassOf` axiom, OWL knows another axiom that can be used to define that the element types $E_2 \dots E_n$ constitute a complete, non-overlapping partition of E_1: `DisjointUnion(`$E_1, E_2, \dots, E_n$`)`.

Because of the different possibilities in UML to specify completeness or non-overlapping of generalization relationships, different cases for the transformation UML $\rightarrow$ OWL can be identified:

- general case
- non-overlapping generalization
- non-overlapping and complete generalization
- complete (but not non-overlapping) generalization
- generalization of data types—treated in the section on data types

General case: The concepts of specialization and generalization in UML and OWL are very similar. If E' is a specialized sub-type of an element type E and i is an instance resp. individual, $E'(i) \rightarrow E(i)$ holds true in both cases. Therefore, the transformation is quite simple. For every generalization relationship "E is generalization of E' (resp. E' is specialization of E)", the axiom `SubClassOf(` E' E `)` is added to the ontology.

Non-overlapping generalization: A generalization is non-overlapping (disjunct), if an instance of a sub-type must not be an instance of another sub-type of the same generalization at the same time. This restriction can be enforced by adding a `DisjointClasses` axiom (instance of the OWL meta class `DisjointClasses`), which contains all sub element types of the generalization.

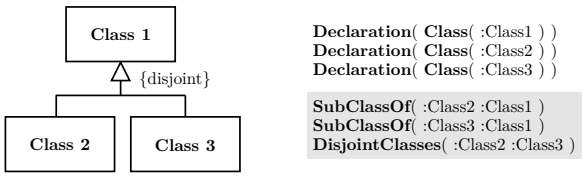


Fig. 3. Example for the transformation of non-overlapping (but not necessarily complete) generalization.

Non-overlapping and complete generalization: If a generalization is non-overlapping and complete, a stronger axiom can be used. A `DisjointUnion(`E E'_1 ... E'_n`)` axioms– which is actually only a shorthand notation—states that every individual, which is an instance of element type E, is an instance of exactly one element type E_i and every individual, which is an instance of element type E_i, automatically belongs to the population of E.

Complete generalization: The situation is similar when the generalization is complete, but not overlapping. In this case, the `DisjointClasses` axiom contained in the axioms combined in the shorthand notation of a `DisjointUnion` axiom is not necessary. This results in the solution shown in Figure 5, which uses an `ObjectUnionOf`-CE and a `EquivalentClasses` axiom.

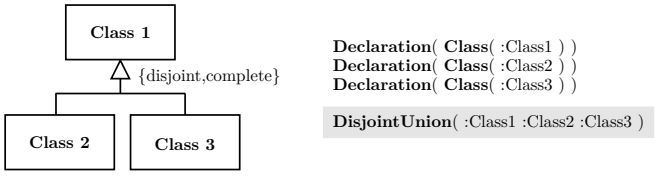


Fig. 4. Example for the transformation of a non-overlapping and complete generalization.

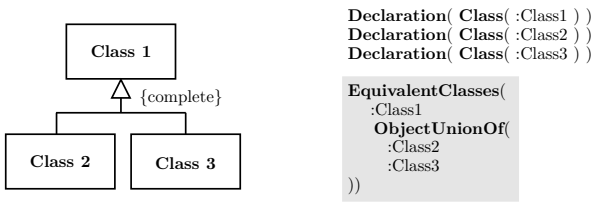


Fig. 5. Example for the transformation of a complete but not non-overlapping generalization.

1) ObjectUnionOf: An `ObjectUnionOf(`$E_1 \dots E_n$`)` defines an element type with a population consisting of those individuals that are instance of one or more element types $E_1 \dots E_n$. If $E_1 \dots E_n$ can be transformed into UML classes $C_1 \dots C_n$, the `ObjectUnionOf` can be represented as abstract class.

Between the abstract class and each member of the union $C_1 \dots C_n$ generalization relations (instances of the UML meta class *Generalization*) must be created during the transformation process. The generalizations are combined into an instance of the UML meta class *GeneralizationSet*. It is marked as *complete*. This is possible, because in UML the definition of a union of element types $E_1 \dots E_n$ is semantically equivalent to an abstract element type and the specification of subtypes.

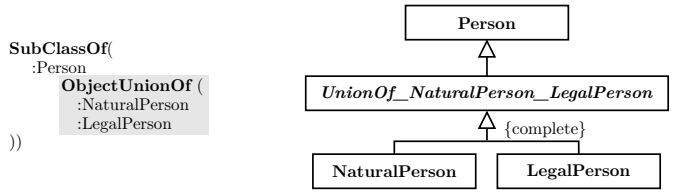


Fig. 6. Example for the transformation of an instance of the OWL meta class `ObjectUnionOf` into an abstract class (instance of the UML meta class *Class*) and inheritance relationships with *GeneralizationSet*.

DisjointUnion: The `DisjointUnion` axiom is a syntactical shortcut. The axiom `DisjointUnion(`A B_1 ... B_n`)` is semantically equivalent to the three axioms

`SubClassOf(A ObjectUnionOf(`B_1 ... B_n`))`
`SubClassOf(ObjectUnionOf(`B_1 ... B_n`) A)`
`DisjointClasses(`B_1 ... B_n`)`

In this particular case, the above mentioned problems of `SubClassOf` axioms and sufficient conditions for class membership can be circumvented and the semantics of the expression can be transformed. The long notation of a `DisjointUnion` axiom contains a `ObjectUnionOf(`B_1 ... B_n`)` CE. As described above, this CE is transformed into $n + 1$ UML classes, n generalization relations and an instance of

the UML meta class *GeneralizationSet* which is marked as *complete*. However, in this case the superclass is not abstract. A name is assigned to this class that corresponds with the IRI of the OWL class. Additionally, the instance of the UML meta class *GeneralizationSet* is marked *disjoint* to reflect the disjointness of the classes $B_1 \ldots B_n$.

F. Intersection

An element type can be defined by the intersection of other element types. An object is an instance of that element type if it is an instance of a set of other element types.

Since UML has no model element similar to *GeneralizationSet* for specializations, the semantic of this construct cannot be expressed completely. If it is possible to transform the element types $E_1 \ldots E_n$ that are part of the `ObjectIntersectionOf`-CE into UML classes $C_1 \ldots C_n$, it is only possible to define an abstract class, which is a subclass of the classes $C_1 \ldots C_n$. Since UML does not allow an automatic classification by sufficient conditions, it is not possible to enforce that an object, which is a instance of all classes $C_1 \ldots C_n$, is also an instance of the abstract subclass.

IV. Data Types

In general, a datatype consists of three components: the value space, the lexical space, and a well-defined mapping from the lexical into the value space. The value space is the—possible infinite—set of values that can be represented by the datatype. The lexical space describes the syntax of the datatype's values. The mapping is used to map syntactically correct values to elements of the value spaces. It is possible that many, even infinite many, syntactically different values are mapped to the same element of the values space.

Primitive datatypes do not have an internal structure. Examples of primitive types are character strings, logical values, and numbers.

Enumerations are a special kind of datatypes with no internal structure. In contrast to general primitive types the lexical space and the value space of an enumeration are equal-sized, well-defined finite sets. The mapping from lexical to value space is a one-to-one mapping. An example for an enumeration datatype are the English names of the days of the week which consist of seven possible values.

In contrast to primitive data types, **complex data types** have an internal structure. The following are examples for complex data types:

- a person's name consisting of a given name and a family name
- a physical measurement consisting of value and unit of measurement
- an address consisting of street name, house number, postal code and city name

Generalization of datatypes can be defined similarly to the generalization of element types. If a datatype A generalizes a datatype B each date that is instance of B (i.e., its lexical representation belongs to the lexical space of B and its value belongs to the value space of B) is also instance of datatype A. For example the integers generalize natural numbers. Each natural number is also an integer.

A. Representation in UML

Apart from a few pre-defined primitive types UML allows the definition of additional datatypes in class diagrams. These can be primitive types, complex datatypes, and enumerations. In UML, datatypes—similar to classes—can have owned attributes (as well as operations which are not discussed here). Therefore, they can be used to describe structures. Figure 7 shows examples for the three kind of datatypes.

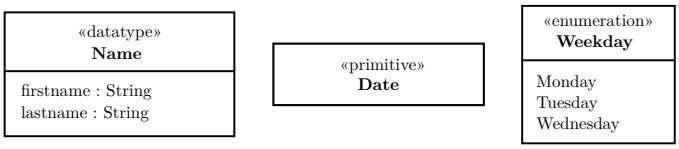


Fig. 7. Examples for datatypes in UML. Left: user-defined datatype with two components. Center: user-defined primitive datatype. Right: Enumeration with three allowed values.

In contrast to instances of classes, "any instances of that data type with the same value are considered to be equal instances."[6, p. 63] Although the graphical representations of datatypes in general (instances of *DataType*) as well as primitive types (instances of *PrimitiveType* and enumerations (instances of *Enumeration*) in particular look similar to the representation of classes (instances of *Class*), they are different elements of the meta model as shown in Figure 8.

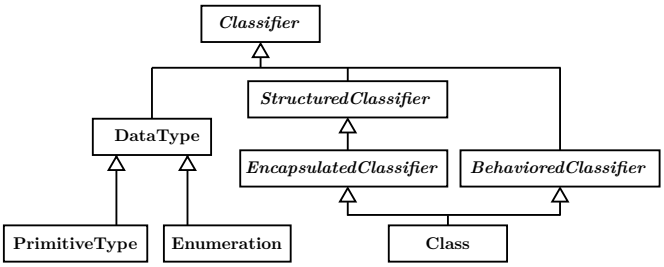


Fig. 8. Extract from the UML meta model, showing the difference between classes and datatypes.

In UML, generalizations are defined for *Classifier* and therefore also for *DataType*. Thus, inheritance/generalization relations between datatypes can be defined in a UML class diagram.

B. Representation in OWL-2

In OWL-2, three different kinds of datatypes can be distinguished:

1) `rdfs:Literal` as base datatype
2) datatypes of the OWL-2 datatype map, which is basically a subset of the XML Schema datatypes [7].
3) datatypes that have been defined within an ontology using `DatatypeDefinition`

The value space of the base datatype `rdfs:Literal` is the union of the value spaces of all other datatypes. The OWL-2 datatype map adopts the value space, lexical space, and the restrictions for user-defined datatypes from

the XML Schema specification. Sets of values (instances of datatypes)—so called *Data Ranges*—can be defined by combining datatypes via common set-theoretic operations. A set of values consisting exactly of one pre-defined list can be described by using `DataOneOf`. A `DatatypeRestriction` allows to define a set of values by restricting the value space of a datatype with *constraining facets*. The OWL-2 datatype map defines which restrictions are allowed. For example a number datatype can be restricted by: less equal, greater equal, equal, and greater.

An OWL-2 datatype is defined by assigning an Internationalized Resource Identifier (IRI) to a `DataRange` using a `DatatypeDefinition` axiom. According to the OWL-2 DL specification this IRI must have been declared to be the name of a datatype.

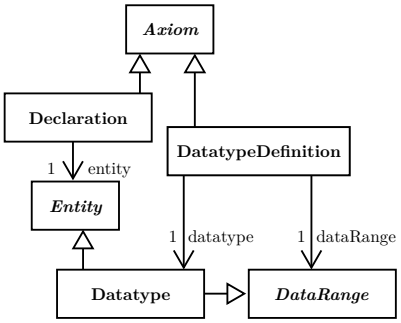


Fig. 9. Extract relevant for datatypes from the OWL-2 meta model.

The abstract syntax (see Figure 9) shows that a datatype is linked indirectly (via an instance of `DatatypeDefinition`) to its value space (an instance of a subclass of `DataRange`. Therefore, it is possible to use a datatype with no assigned value space. By definition this datatype has the value space of `rdfs:Literal`.

Subclasses of `DataRange` (e.g., `DataUnionOf`), which are used for the definition of value sets (and therefore datatypes), have references to `DataRange`. `Datatype` is a subclass of `DataRange`, too. Thus, arbitrarily nested constructions of datatype-defining elements are possible.

C. Primitive Types

Three cases have to be considered for the UML → OWL-2 transformation of primitive types:

1) The datatype is one of the four pre-defined datatypes "Boolean", "Integer", "String", or "UnlimitedNatural".
2) The datatype is one of the XML Schema datatypes.
3) The definition of the (user-defined) datatype is part of the UML-model.

Since OWL-2 uses the datatype-definitions from XML Schema, a datatype in case (1) can be transformed into its corresponding datatype from XML Schema. Primitive types can be recognized by the fact that they are contained in a package "UMLPrimitiveTypes".

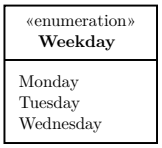
The transformation in case (2) is even more obvious because a datatype is used that is also present in OWL-2. The name of the package containing the primitive types depends on the UML type library used. A common package name is "XMLPrimitiveTypes". This name can be used to recognize primitive types that fall under case (2). The XML Schema datatype can be referenced in the ontology by adding the XSD namespace to the type's name.

For user-defined datatypes in case (3), a new datatype is defined in the ontology by using a `Datatype` axiom. OWL-2 datatypes—like all OWL-2 model elements—are identified by unique IRIs. Therefore, an appropriate IRI must be generated during the transformation. In UML, elements (including datatypes) are uniquely identified by their name and package hierarchy. Therefore, a combination of package and datatype name can be used for the IRI.

For the transformation OWL-2 → UML, primitive types are difficult. OWL-2 offers a variety of possibilities to define new datatypes. However, some primitive types—and probably the most common ones—can be transformed. The primitive types of OWL-2 derive from the XML Schema datatypes. There are established UML-libraries for the XML datatypes. Therefore, it is sufficient to include such a library into the transformation process. An instance of a primitive type contained in the library can be looked up by the IRI of the OWL-2-datatype and references as necessary.

D. Enumerations

As mentioned in Section VIII, several authors have already discussed how to transform enumerations. In OWL-2 the data range `DataOneOf` is suitable to define a datatype with a fixed pre-defined value space. Each lexical value of the `DataOneOf` data range is transformed into an *EnumerationLiteral* instance and vice-versa. OWL-2 as well as UML support the specification of datatypes for the elements of an enumeration: An OWL-2 *Literal* instance has a *datatype* attribute, an UML *EnumerationLiteral* instance has a *classifier* attribute referencing the datatype.




```
Declaration( DataType( :Weekday ) )

DatatypeDefinition(
  :Weekday
  DataOneOf( "Monday" "Tuesday" "Wednesday" )
)
```

Fig. 10. Example for the transformation of an enumeration.

For the transformation OWL-2 → UML one has to consider the fact that in OWL-2 the data range `DataOneOf` can be used without a `DatatypeDefinition` which assigns a name to it. Since an UML *Enumeration* necessarily needs a name, it can be generated based on the literals contained in the data range.

E. Complex Data Types

OWL-2 datatypes consist of exactly one literal and are therefore not further structured. Since OWL-2 is built upon the Resource Description Framework (RDF), there is the theoretical possibility to use a blank node and the RDF-instruction `parseType="Resource"` to implement complex data as shown in this listing:

```
<rdf:RDF xml:base="http://example.com/persons/"
```

```
   xmlns="http://example.com/persons/"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

   <owl:Ontology rdf:about="http://example.com/persons/"/>

   <owl:Class rdf:about="Person" />

   <owl:NamedIndividual rdf:about="Timmi">
      <rdf:type rdf:resource="Person"/>
      <hasName rdf:parseType="Resource">
         <first>Timmi</first>
         <last>Tester</last>
      </hasName>
   </owl:NamedIndividual>
</rdf:RDF>
```

However, neither the OWL-1 nor the OWL-2 specification mention `parseType="Resource"`. Therefore, it is probably not a valid construct for OWL-2. Even if this notation was valid for OWL-2 and an element type could be assigned to such an anonymous individual, the definition of the element type would be indistinguishable from the definition of a "normal" element type.

The UML $\rightarrow$ OWL-2 transformation of complex datatypes, i.e., datatypes with owned attributes, is similar to the transformation of UML classes with owned attributes into OWL-2 classes and properties. There are two characteristics of UML datatypes that have to be considered:

1) Values do not have an identity.
2) Every value exists only once.

Since the transformation is similar to the transformation of classes the instances of the resulting element in the ontology will be individuals. In OWL-2, every (typed) individual must have a name. Therefore, the semantics for characteristic (1) is changed: in UML, the instance of the datatype does not have an identity. The corresponding individual in OWL-2 is assigned with an IRI by which it can be referenced (and also identified).

Characteristic (2) requiring that every value must exist not more than once, can be ensured by using `HasKey` axioms. For every UML datatype D with owned attributes $a_1 \dots a_n$ that is transformed into a OWL-2 class C with data property $dp_1 \dots dp_n$, the following axiom is added to the ontology:

`HasKey(` C `()` `(` $dp_1 \dots dp_n$ `)` `)`

This axiom ensures that every occurrence of an individual with the same values for $dp_1 \dots dp_n$ is one and the same individual.

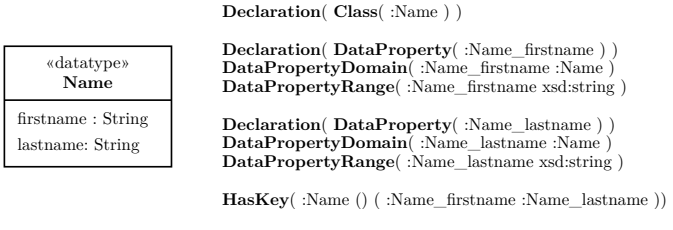


Fig. 11. Example for the transformation of a complex datatype.

F. Generalization of datatypes

In general, the transformation of a datatype generalization in a UML class diagram is not possible, since OWL-2 has no support for inheritance/generalization of datatypes. In the special case of a complete generalization of datatypes with no internal strcuture (e.g., enumerations), a transformation is possible: While the generalization of UML classes can be transformed into an OWL-2 `ObjectUnionOf` class expression, this is not possible for datatypes. As the name suggest, an `ObjectUnionOf` can only be used for classes. Instead, an instance of `DataUnionOf` is used. The sub-datatypes combined in the `DataUnionOf` constitute a new data range. By using a `DatatypeDefinition` axiom a name is assigned to this set of datatypes. This name is the name of the super-datatype from UML. Figure 12 shows an example for such a transformation.

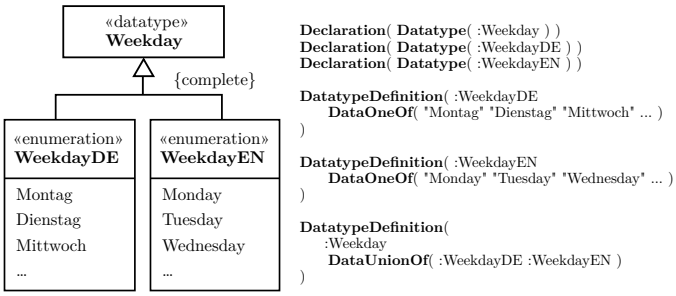


Fig. 12. Example for the transformation of a generalization relation between datatypes.

V. Relationship Types

In a software system one can typically find a variety of relations between instances of the element types. The characteristics of these relations are described by means of relationship types. Instead of the term "relationship type" the term "relation" is often used. This is not quite correct, as a relation refers to a concrete instance of a relationship type between instances of element types. A relationship type describes which characteristics apply to all of its relations in general.

A relation includes participants (=participating instances of an element type) that play a certain role. Applied to relationship types, these participants are element types that play a certain role within the relationship type. It is possible to omit the indication of roles for a relation and the relationship type, respectively.

A relationship type with two members is called binary relationship type. Corresponding relations are called binary relations, accordingly. Since arbitrary n-ary relations can be transformed into binary relations [4, Chap. 6][8], only binary relationship types are considered in the following.

In UML, binary relationship types can be presented in two different ways—as associations or as class-dependent attributes. Associations are depicted by lines between element types. The name of the relationship type can be written close to the line, maybe with an arrow indicating the direction. Class-dependent attributes are listed in the middle section of an element type. Although the concrete graphical syntax of associations and attributes differs significantly from each other, both are represented in the abstract syntax by the same UML meta class *Property*. Because of this similarity, it is

useful to consider the transformation of associations and class-dependent attributes together, since they differ only within a few aspects.

OWL knows two different constructs to connect elements:

- Object Properties – for relations between instances of classes
- DataProperties – for relations between an instance of a class and an instance of a datatype.

At the declaration of an OWL property—i.e., an indication that a property with this name exists—initially no information on the related element types E_1 and E_2 is made. Therefore, an instance of such a relationship type can be used for the connection of arbitrary objects. Only by the use of further axioms statements about the domain and the range of the property are made. Thus, the element types E_1 and E_2 are determined.

Since the UML meta class *Association* that represents an association is a sub-type of *Classifier*, all associations are direct components of a UML package. The OWL concept most similar to an association is an object property. Via its declaration it is also a component of the ontology. The transformation of class-dependent attributes is a bit more complicated because there is no obvious corresponding concept to them in OWL. The main issue is that in OWL classes do not contain any other model elements—as it is the case in UML. But again, the connection can be mapped to an object property or a data property. In both cases, the decision on whether an instance of the UML meta class *Property* is mapped to an object property or a data property depends on the kind of element type the *Property* instance points to. If it is an instance of the UML meta class *Class* or a complex datatype—i.e., an instance of the UML meta class *DataType* with dependent attributes—an object property will be used. If however, it is an instance of a simple data type—i.e., an instance of the UML meta classes *PrimitiveDataType* or *Enumeration*—, a data property will be used.

In contrast to UML, where the types of an association (domain and range) are always specified, it is optional to specify domain and range for properties in OWL. Per definition, the domain and range of such properties is `owl:Thing` (respectively `owl:Literal` for the range of a data property). Such properties can be used to connect instances of arbitrary element types. In order to restrict the properties like in an UML model, it is necessary to add appropriate axioms that specify the allowed classes and datatypes for domain and range of the property. The range of an instance of the UML meta class *Property* can be determined as follows: it is the element type that is connected to the *Property* instance via an instance of the UML meta class *Type* appearing in the role *type*. To determine the domain one has to distinguish between associations and class-dependent attributes.

- In case of a class-dependent attribute a connection exists between the *Property* instance and a instance of *Class* in role *class*. The element type represented by this *Class* instance is the wanted type.
- If the *Property* instance is part of an association (i.e., a connection to an instance of *Association* where the role *association* exists), the type of the other member-end of the association has to be chosen.

Figure 13 illustrates this selection of domain and range.

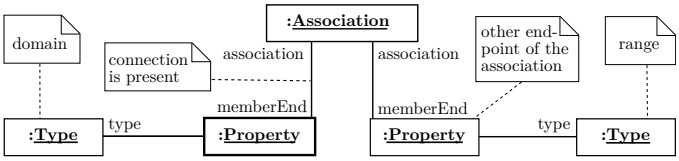


Fig. 13. Selection of the domain and range in case of an association. The focus is on the thicker bordered *Property* instance.

In order to avoid that two OWL properties that are the transformation result of different instance of the UML meta class *Property* are interpreted as a single property, all those properties that are not in an inheritance relationship will be marked as disjoint. For this purpose, `DisjointObjectProperties` and `DisjointDataProperties` axioms are added to the ontology.

As described above, OWL distinguishes between data properties connecting classes with datatypes, and object properties connecting classes with classes. Data properties are mapped onto class-dependent attributes in UML. With a few exceptions object properties are also transformed into class-dependent attributes. Object properties to which there is an inverse relation within the ontology are treated seperately. Such an inverse relation may be specified in several ways: by explicit specification using an `InverseObjectProperties` axiom, by using an anonymous inverse `InverseObjectProperty` or by marking an object property as symmetric or inverse-functional.

It must be expected that more than one class is specified as domain or range of a property. UML does not allow this notation. In such cases, a helper class is added, which inherits from all classes in the domain or range. Figure 14 illustrates such a construction. At those places where the affected object property is used, the new auxiliary class will be used in the UML model, accordingly.

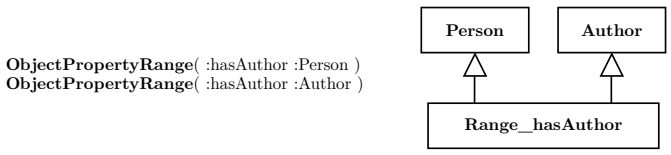


Fig. 14. Example for the transformation of multiple axioms indicating the range of an object property into UML classes with inheritance relations.

One possibility to transform object properties with `owl:Thing` as domain and/or range is to define a single base class C_{super} within the UML model. C_{super} is defined as super-class of all other classes in the model and thus corresponds to `owl:Thing`. In this particular case of a single base class, an object property without definition of domain and range can be mapped to an instance of the UML meta type *Association* with two member ends of type C_{super}.

A. Inheritance

Sometimes it is necessary that from one relationship between two objects, another relationship follows automatically.

In other words, the population of a relationship type R_2 includes the population of another relationship type R_1. All instances of R_2 are automatically also instances of R_1.

In UML, the inheritance of relationship types is realized similarly to the inheritances between element types. If both relationship types are represented by an association, a generalization between these two associations can be created. In the case of class-dependent attributes, a subsetted attribute can be specified. Also, OWL allows to express that two relationship types are in an inheritance relationship. For this purpose, the axioms `SubObjectPropertyOf` and `SubDataPropertyOf` exist. An example of such a transformation is shown in Figure 15.

An inheritance relationship (instance of the UML meta class *Generalization*) between two instances of the UML meta class *Association* can be transformed into OWL by using instances of the OWL meta classes `SubPropertyOf`. Since a bidirectional association is transformed into two object properties, the generalization between two bidirectional associations must be transformed into two instances of the OWL meta class `SubPropertyOf` as well.

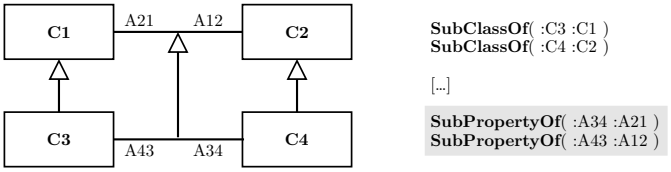


Fig. 15. Example for the transformation of generalized associations.

For the transformation of a `SubObjectPropertyOf` axiom the class-dependent attribute *subsettedProperty* of the UML meta class *Property* can be used. It specifies the parent *Property* instance.

B. Cardinality Constraints

Cardinality constraints are one of the most important types of restrictions for conceptual modelling. They allow a technical limitation of the quantity of relations that an object can have with other objects. In UML, cardinality constraints are called multiplicities and can be specified for both associations as well as class-dependent attributes.

OWL also knows constructs to describe cardinalities. Class expressions (CE) are used to describe certain sets of individuals. In the following, a binary relation $R(x, y)$ between two individuals x and y is considered.

The CE `ObjectMinCardinality( n R )` describes the set of individuals that are in a relation R to more than n individuals: $\{x : |\{y : R(x, y)\}| \geq n\}$. Corresponding CEs exist for sets of individuals that are linked with less than n individuals (`ObjectMaxCardinality`) or exactly n individuals (`ObjectExactCardinality`). If y is a datatype, the CEs `DataMinCardinality`, `DataMaxCardinatlity`, or `DataExactCardinatliy` are used.

`ExactCardinality` is only a shortcut for a pair of `MinCardinality` and `MaxCardinality` elements. However, using this shortcut improves the readability of the ontology. Therefore, a UML cardinality constraint with the value for upper and lower bound is tranformed into an instance of the UML meta class `ExactCardinality` with a equivalent value.

If the value of the upper bound is 1, an additional instance of the OWL meta class `FunctionalObjectProperty` or `FunctionalDataProperty` is created. Again, although this is only a abbreviating notation, the additional axiom improves the comprehensiveness of the ontology, because the type of property is easier to recognize.

However, it is not possible and sufficient to add the corresponding CE for the cardinality constraints directly to the ontology. On the one hand, CEs are not axioms and can therefore not be added to the ontology directly. On the other hand, in UML, cardinality constraints of class-dependent attributes and associations always refer to a specific class. In OWL properties are not directly contained in classes (see above). Therefore, cardinality constraints defined for properties as CE do not affect classes.

These difficulties can be solved by adding an instance of the OWL meta class `SubClassOf` to the ontology for every cardinality constraint. The sub-type is defined as the OWL class that corresponds to the UML class for which the relationship type and its cardinality constraint were defined. The super-type is the CE that represents the cardinality constraint. Thus, the cardinality constraints of the CE are inherited by the class that is related to the association or the class-dependent attribute.

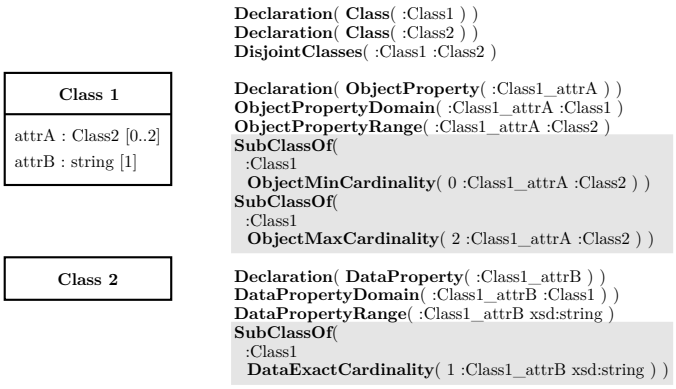


Fig. 16. Example for the transformation of klass-dependent attributes and associations with cardinality constraints.

For the direction OWL $\rightarrow$ UML it has to be considered that the CE `ObjectMinCardinality` and `ObjectMaxCardinality`—and those for data properties, respectively—define anonymous element types that specify restrictions on the occurrence of a property. If an anonymous element type, defined by such cardinality constraints is used as superclass C_p within an `SubClassOf`$(C_c\ C_p)$ axiom, the cardinality constraints become constraints of the class-dependent attributes of the subclass C_c. Figure 17 illustrates this case.

OWL offers two axioms to characterize properties as functional. However, both axioms are only syntactic shortcuts for a subclass axiom and a cardinality CE. Therefore, they can be transformed into cardinality constrains 0..1 of the corresponding instance of the UML meta class *Property*. The `InverseFunctionalObjectProperty` axiom to char-

```
Declaration( Class( :Book ) )
Declaration( Class( :Person ) )
Declaration( ObjectProperty( :author ) )
Declaration( DataProperty( :title ) )

SubClassOf( :Book
   ObjectMinCardinality( 1 :author :Person ))
SubClassOf( :Book
   ObjectMaxCardinality( 5 :author :Person ))
SubClassOf( :Book
   DataExactCardinatliy( 1 :title xsd:string ))
```

Book
author : Person [1..5]
title : string [1]

Fig. 17. Example for the transformation of various cardinality restricting axioms into cardinality constrains of class-dependent attributes.

acterize an object property as inverse-functional is also a syntactic shortcut. However, the problems with `SubClassOf` axioms and sufficient conditions for class membership mentioned above prevent a similar transformation. The restriction can be preserved by mapping the OWL property onto one member-end of an instance of the UML meta class *Association* and by setting a cardinality constraint 0..1 for the other member-end of that association.

C. Value constraints

In case of a value restriction, the second participant of a relationship type is set to exactly one value or an object. In UML, it can be defined that a class-dependent attribute is pre-set to a fixed value during the instantiation. In order to prevent the change of this value even during dynamic use, this value can also be marked as immutable. For both, object properties as well as data properties, OWL offers the possibility to set the second participant to a fixed value. The value constraint can also be achieved by defining a one-element element type and a corresponding cardinality axiom. However, with `ObjectHasValue` and `DataHasValue`, OWL offers shortcuts, which make the semantics clearer to a human reader.

Similar to the transformation of class-dependent attributes with cardinality constraints into `SubClassOf` axioms of the containing class, as a value constraint is transformed into a `SubClassOf` axiom as well. The super-type will be an instance of the OWL meta class `DataHasValue` (a data range). Within that data range, the fixed value and the data property generated from the class-dependent attribute are defined.

In the transformation OWL $\rightarrow$ UML, one can use the fact that if an element type defined by a `DataHasValue` CE is used as super-type C_p within `SubClassOf(`C_c C_p`)` axiom the fixed value becomes a necessary condition for instances of the class C_c. Every single instance of that class will have exactly this fixed value. In a UML model, this value can be defined for an instance of the UM meta class *Property* by setting the class-dependent attribute *default*.

```
Declaration( Class( :Book ) )
Declaration( DataProperty( :title ) )
Declaration( DataProperty( :printed ) )

SubClassOf( :Book
   DataExactCardinatliy( 1 :title xsd:string ))
SubClassOf( :Book
   DataHasValue( :printed "true"^^xsd:boolean ))
```

Book
title : string [1]
printed : boolean = true

Fig. 18. Example for the transformation of an axiom for a value constraint into a value constraint of a class-dependent attribute.

D. Part-Whole-Relationships

Part-whole relations (also known as composition and aggregation) are a special kind of relationship types. They play an important role in modelling [4, p. 137]. They can be used to express a certain semantic of the relationship. Additionally, further restrictions can be imposed on the respective relationship type. A part-whole relation is antisymmetric—i.e., if T is part of G, G can not be part of T. There is disagreement on whether part-whole relations are transitive or not [4, p. 142]. Since a part-whole relation is a binary relationship type, the previously made statements and observations for general relationship types also apply to this kind of relationship types.

UML knows two kinds of part-whole relations: aggregation and composition. They differ in both, (graphical) syntax and their semantics. An aggregation is anti-symmetric and transitive. Objects linked by it form an acyclic graph [9, p. 171]. It is allowed that a "part" is part of more than one "whole". Composition is a stronger form of aggregation. In addition to anti-symmetry and transitivity, a "part" may—at a time—be only part of a single ''whole". Futhermore, the existence of a "part" depends on the existence of the "whole". It can not exist without something it is part of. OWL has no special constructs to identify part-whole relationships.

Like other associations between two classes, part-whole relation types are transformed into object properties. Moreover, the additional restrictions mentioned above are taken into account:

a) aggregations are antisymmetric

b) an object must not be in an aggregation relation to itself—that would be a contradiction to the antisymmetry,

c) an object must not be part of more than one composition,

d) an instance of a class that is part of a composition must not exist alone.

The asymmetry can be achieved by adding an `AsymmetricObjectProperty` axiom to the object property that has been transformed from the association with aggregation or composition characteristic.

Restriction (b) can be transformed to OWL by adding an `IrreflexiveObjectProperty` to the ontology for each association with aggregation or composition characteristic. This axiom prohibits the use of the corresponding object property to connect an individual with itself.

Restriction (c) can be achieved by adding a `FunctionalObjectProperty` or an `InverseFunctionalObjectProperty` axiom. If the association with composition characteristic is bidirectionally navigable, it makes no difference what type of axiom is used. However, if the association is only navigable from one direction the following distinction has to be made. If the association is navigable from "part" to "whole", a `FunctionalObjectProperty` is used. A connection between an individual of the "part"-class to more than one individual of the "whole"-class would make the ontology inconsistent. An `InverseFunctionalObjectProperty` axiom is used if the association is navigable from "whole" to "part".

The enforcement of a constraint of the form (d) is not possible, since the open world assumption is used for OWL. The individual in question could be part of a composition that is not explicitly listed in the ontology.

E. Inverse

If a conceptual model contains an relationship type $R(E_1, E_2)$ with two participating element types E_1 and E_2, a common wish is to have the choice to use instances of both element types as first or second participant. To make this possible, one can define an inverse relationship type $R_{inv}(E_1, E_2)$.

For example, consider a book containing chapters. The relationship type $contains(Book, Chapter)$ is defined for the element type $Book$ and $Chapter$. Instances of type $Book$ must always appear as the first participant of such a relationship. If a relationship type $isContainedIn(Chapter, Book)$ that is inverse to $contains$ is defined, statements equvialent to those with $contains$ can be made with an instance of type $Chapter$ as first participant.

Although UML provides no possibility to explicitly mark two arbitrary associations as inverses of each other, the two ends of a binary bidirectional association can be seen as two inverse relations.

The definition of inverse relationship types is only possible for object property, not for data properties. A value (an instance of a datatype) must not contain properties itself. OWL offers three possibilities to use inverse relationship types:

1) An `InverseObjectProperties` is used to declare two previously defined object properties as inverses of each other..
2) The inverse of an object property can be used directly by using the object property expression `ObjectInverseOf` without assigning a name to it.
3) An object property is marked as inverse-functional.

It should be noted that for two individuals a and b the inverse object property $op_{inv}(b, a)$ is automatically part of the ontology if $op(a, b)$ is contained in the ontology.

During the transformation UML $\rightarrow$ OWL bi-directional associations are transformed into two—initially independent—object properties. However, such associations are equivalent to two directed mutually inverse associations. [9, p. 165] Therefore, an instance of the OWL meta class `InverseObjectProperties` is created and linked with the corresponding instances of `ObjectProperty`.

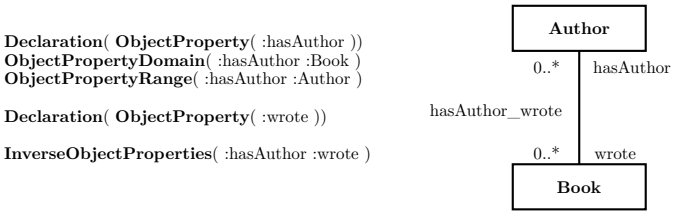


Fig. 19. Example for the transformation of an inverse object property into an instance of the UML meta class *Association*.

When transforming inverse object properties in OWL ontologies into UML no class-dependent attributes are used—in contrast to the generic case described above. Otherwise, the connection between the two links could not be seen. Instead, an instance of the UML meta class *Association* is used as transformation target. The two instance of the UML meta class *Property* occurring as member-ends are mutually set as the value of the *opposite* attribute.

VI. Constraints

Some very common constraints have been discussed in the previous sections, such as cardinality constraints or non-overlapping generalizations. In this section, further restrictions on element types and relationship types will be discussed, namely keys for element types and "conditional relationship types" that work on a combination of element and relationship types.

A. Key Constraints

Key constraints can be used to enforce that there are no two different instances of an element type for which all relations specified in the key have an identical value, or point to the same object. There are simple keys that are based on only one relationship, as well as composite keys, which are based on multiple relations.

UML offers the possibility to define a single key per element type. Class-dependent attributes (instances of the meta class *Property*) can be marked that are part of this key. These marked attributes can be used to identify an instance of the element type.

OWL offers the `HasKey` axiom to define composite keys. Such a key can not only be defined for named element type but also for any CE. The relationship types to be considered are divided into two sets, the object properties and the data properties. With the axioms `FunctionalObjectProperty` and `FunctionalDataProperty` OWL provides yet another way to define especially strong simple keys. An identity is defined independently of the element type of the object, on the basis of the relationship alone.

To transform a UML class with a key, i.e., some of its class-dependent attributes are marked with `isID=true`, a corresponding instance of the OWL meta class `HasKey` is added to the ontology.

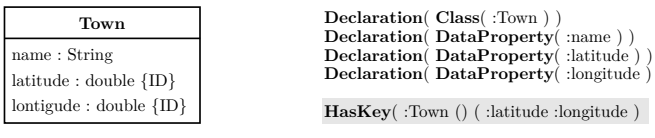


Fig. 20. Transformation of a composite key constraint.

The information that the properties appearing within `HasKey` axioms of an OWL ontology form a key can be transformed into an UML model by setting the value of the class-dependent attribute *isID* for the instances of the UML meta class *Property* that have been generated from those properties. The transformation of functional object and data properties has already been discussed above.

B. Conditional relationship types

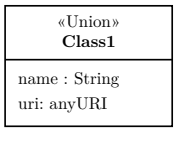
A conditional relationship type consists of a set of relationship types. An object must not appear more than once as a member of an instance of these relationship types. As an example, consider an address which might include either a visiting address or a postbox, but not both.

UML does not provide a special construct to defined such conditional relationship types. However, the ISO 19100 "UML profile" defines a new meta class *Union* with the desired semantics. Only one of a *Union*'s properties must be used. In an UML diagram an instance of a *Union* is depicted by adding the stereotype «Union» to a class symbol. However, this is not a "real" UML stereotype, as the semantics of the model element is changed. The set of the *Union*'s properties is defined by the set of class-dependent attributes.

Two different mappings have been developed to transform an instance of the meta class *Union* into OWL. The first mapping is only valid if the types of all class-dependent attribute are ether datatypes or classes— but not a mixture of both. In that case the attribute are transformed into object properties or data properties. By contrast, the second mapping also allows the transformation of a mixture of classes and datatypes. However, a larger number of axioms is required to reproduce the semantics.

Mapping 1: Let C be a class representing an instance of *Union*. Let $p_1 \dots p_n$ be its properties. It must be ensured that only a single property $p_x \in p_1 \dots p_n$ is specified for an individual. To achieve this, a helper property p_{Union} with domain C and the axioms $p_i \sqsubseteq p_{Union} \forall i \in 1..n$ are added to the ontology.

A `DataExactCardinality` axiom is used to restrict the number of p_{Union} properties for each individual of class C to exactly one. This prevents the setting of two or more different properties. Due to OWL's OWA it cannot be guaranteed that a property has been specified explicitly at all. This problem has been discussed above in the section on cardinality constraints.



Declaration(**Class**(:Class1))

Declaration(**DataProperty**(:name))
DataPropertyDomain(:name :Class1)
DataPropertyRange(:name xsd:string)
SubDataPropertyOf(:name :Class1_UnionProperty)

Declaration(**DataProperty**(:uri))
DataPropertyDomain(:uri :Class1)
DataPropertyRange(:uri xsd:anyURI)
SubDataPropertyOf(:uri :Class1_UnionProperty)

Declaration(**DataProperty**(:Class1_UnionProperty))
DataPropertyDomain(:Class1_UnionProperty :Class1)

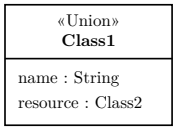
SubClassOf(
:Class1
DataExactCardinality(1 :Class1_UnionProperty))

Fig. 21. First approach for a transformation of an ISO 19100 Union.

Mapping 2: For each property $p_i \in p_1 \dots p_n$ of the *Union* an OWL helper class C_i is defined. By using a `Disjoint-Classes` axiom, it is stated that these n classes are disjoint in pairs. For each class, it is additionally stated that it is equivalent to the set of those individuals that are connected to exactly one individual or literal via p_i:

`EquivalentClasses(` C_i
`DataExactCardinality( 1` p_i `) )` bzw.
`EquivalentClasses(` C_i
`ObjectExactCardinality( 1` p_i `) )`

By using the first mapping only $(n+3)$ per UML property will be added to the ontology. The second mapping requires $(2n+1)$ additional axioms per property. Therefore, it is smart to use the first option if an instance of the meta class *Union* is composed exclusively of data types or classes, and to use the second option only when a mixture of both is present.



Declaration(**Class**(:Class1))
Declaration(**Class**(:Class2))

Declaration(**DataProperty**(:name))
DataPropertyDomain(:name :Class1)
DataPropertyRange(:name xsd:string)

Declaration(**ObjectProperty**(:resource))
ObjectPropertyDomain(:resource :Class1)
ObjectPropertyRange(:resource :Class2)

Declaration(**Class**(:Union_Class1_name))
EquivalentClasses(
:Union_Class1_name
DataExactCardinality(1 :name))

Declaration(**Class**(:Union_Class1_resource))
EquivalentClasses(
:Union_Class1_resource
ObjectExactCardinality(1 :resource :Class2))

DisjointClasses(:Union_Class1_name
:Union_Class1_resource)

Fig. 22. Second approach for a transformation of an ISO 19100 Union.

VII. Evaluation

This section deals with the question of whether the transformation rules presented in the previous sections are correct and—within their previously specified limits—complete. For this purpose, three different kind of analysis were conducted:

1) Coverage of the meta models
2) Analysis of individual transformation rules
3) Check the transformations automatically

Due to space limitations, only the third analysis is presented in detail.

One advantage of using QVT-R for the transformation is the generation of so-called "trace classes" and their instances during the execution of the transformation rules. Instances of the trace classes depend on the input models. In contrast, the trace classes itself are independent of the processed models. They are determined only by the transformation rules and the meta models. These recordings are used for tests 1) and 2).

Test 1 deals with the coverage of the meta models. It shows which part of the UML and OWL meta models is captured by the transformations at all. Further investigations are necessary, as an examination of the coverage of the meta-models is not sufficient for the evaluation of transformations. Even a complete coverage of the meta models cannot guarantee semantically preserving transformations. Such a complete coverage could actually be achieved by trivial and meaningless transformations. Consider the following example: all element types of the meta model M_1 are mapped to a single element type B of the target meta model M_2. Thus, a complete coverage is achieved for M_1. To also achieve complete coverage

for the element types of M_2, a second transformation rule is needed. For each instance of the element type A in meta model M_1 it creates an instance of every element type in meta model M_2. As a result a complete coverage of of M_2 is also achieved.

Therefore, it is necessary to investigate the transformation rules further. For this purpose, individual, mutually inverse transformation rules with their mutual dependencies and the dependencies to the meta element types are analysed in **test 2**. Rules that only artificially increase the coverage of the meta models would be detected by this test. Such a rule would attract attention because

- it is connected to instances of unusual many element types or
- it creates instances of unusual many element types.

Due to the complexity of a manual analysis of the transformation rules and the risk to overlook errors, an automatic verification of transformations is desirable. Such a verification is presented by **test 3**.

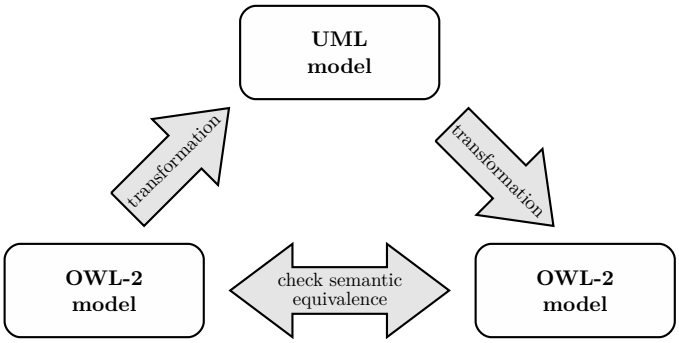


Fig. 23. Procedure for checking the correctness of the transformations.

Figure 23 shows a sketch of how to show the correctness of the transformation rules for certain parts of the meta model. The transformation rules are executed in both directions. After that input model and output model are compared *appropriately*.

It is advantageous to use an OWL ontology as input model (and thus also as output model). During the following comparison, available software tools such as reasoners can be used. The following shows how an " appropriate" comparison might look like.

Set U be the UML meta model, O the OWL meta model, u a model conforming to U, and o a model conforming the O. Let $\vec{T}_{UO}$ and $\vec{T}_{OU}$ be the transformations UML $\rightarrow$ OWL respective OWL $\rightarrow$ UML described above. Ideally, the consecutive execution of the transformations $o_2 = \vec{T}_{UO}(\vec{T}_{OU}(o_1))$ should create an ontology such that o_1 and o_2 are *semantically equivalent*.

What does *semantically equivalent* mean? It can be observed that there are models M_1 and M_2 with a different structure, for which each instance m that conforms to M_1 also conforms to M_2. Thus, the models describe the same (static) semantics. Similarly, models can be found that have the same informational content and differ only by the names of the elements. With a simple renaming, any instance that conforms to the first model, can be be transferred to an instance conforming to the second model. Overall, the checking for semantic equivalence can be cut down to the question of whether a *Total Ontology Mapping* [10] exists between both ontologies.

An ontology is a pair $O = (S, A)$ with S being the signature of the ontology and A its set of axioms. The signature describes the vocabulary used within the ontology. The set of axioms describes how the elements of S are put into relation.

A Total Ontology Mapping between an ontology $O_1 = (S_1, A_1)$ and an ontology $O_2 = (S_2, A_2)$ is a morphism $f : S_1 \rightarrow S_2$ that maps both signatures of the ontology in a way such that $A_2 \models f(A_1)$. All interpretations that satisfy the axioms of O_2 also satisfy the renamed axioms of O_1.

A. Computation of a Total Ontology Mapping for OWL-2

In OWL, the *signature* of the ontology is formed by the instances of the meta class `Entity`. The elements of the signature are divided into disjoint sets `Class`, `ObjectProperty`, `DataProperty`, `AnnotationProperty`, `Datatype`, `NamedIndividual`. Thus, the signature has the form$S = (S_C, S_{OP}, S_{DP}, S_{DT}, S_I)$. Annotations are ignored as they do not carry any semantic information. Since the sets are disjoint, the search for renaming can be restricted to one set. That significantly reduces the complexity of the search.

For simplicity, it is assumed that S_1 only contains elements that are used in A_1 and S_2 only contains elements that are used in A_2. Otherwise, unused items can be deleted without changing the statement of the axioms.

It is further assumed that the components of the signatures of the two ontologies have the same size: $|S_{X1}| = |S_{X2}|$, $X \in \{C, OP, DP, DT, I\}$. If this is not the case, an appropriate amount of previously unused elements is added to the smaller set.

In order to maintain a clear notation, only the subsets S_{C1} and $S_{C2}$for the classes are considered in details. The other four subsets S_{OP}, S_{DP},S_{DT}, and S_I are handled similarly.

The algorithm works as follows. Put the elements of S_{C1} and S_{C2} into an arbitrary order. The result are two ordered lists $S_{C1} = (c_1, \dots, c_n)$ and $S_{C2} = (d_1, \dots, d_n)$. For all possible permutations $\sigma_C : N \rightarrow N$, $N = \{1, \dots n\}$ check if every axiom $a \in f(A_1)$ can be inferred from A_2 with $f : (S_{C1}, \dots) \rightarrow (S_{C2}, \dots)$ and $\sigma_C(c_i) = d_i \ \forall i \in \{1, \dots n\}$. If such a permutation can be found, a Total Ontology Mapping between the ontologies O_1 and O_2 exists.

The procedure described in the previous paragraphs:

1) Apply the transformation $\vec{T}_{OU}$ to the input ontology o. The result is $m = \vec{T}_{OU}(o)$.
2) Apply the transformation $\vec{T}_{UO}$ to the UML model m. The result is $o' = \vec{T}_{UO}(m)$.
3) Use the algorithm to test if a Total Ontology Mapping between o and o' exists.
4) Use the algorithm to test if a Total Ontology Mapping between o' and o exists.

can be applied in instances of single meta classes or an arbitrary combination of meta class elements.

VIII. Related work

Two fundamentally different approaches for a transformation between UML and OWL-2 can be identified: XML-based transformations and transformations that are not based on XML.

A. Transformations based on XML

All XML-based approaches have a number of disadvantages in common. When working with documents containing a serialization of a model in concrete syntax only one model level is visible. Usually, only the names of meta models elements are available. The internal structure of the meta model and internal connections are inaccessible. XML-based transformations that use XML Metadata Interchange (XMI) documents and/or ontologies written in XML-based syntaxes of OWL and RDF lead to further problems. For example, the sequence of XML elements in two different serialization of one model can be almost completely different. It is easy to see that this leads to unnecessarily complex transformation rules. Besides others, [11][12] point out these problems as well.

Cranefield has addressed the connection between UML and ontologies in two articles: Cranefield and Purvis have examined how the UML and the Object Constraint Language (OCL) can be used to model ontologies [13] in general. The objective in this early work was not the transformation from UML to OWL, but rather the use of UML as an ontology modelling language. A transformation from UML class diagrams into Java code as well as into RDF-Schema is presented by Cranefield in a later article [14].

Falkovych presents a transformation of UML models into DAML + OIL (a predecessor of OWL) and RDFS using XSLT [15].

Gaševi Djuri et al. describe the transformation of a UML class diagram into an OWL ontology by using XSLT [16]. In the creation of the class model, a special UML profile "Ontology UML Profile"—defined by Djurić et al. in [17]—must be used.

Leinhos describes two variants for the transformation of UML models into OWL ontologies [18]. The UML models are serialized as XMI files. For the OWL ontologies the RDF/XML syntax is used. In one variant, specially constructed UML class diagrams are transformed into OWL ontologies. In the other variant, elements are added to the ontology that are not present in the original UML model and that do not match the semantics of the UML model.

B. Transformations not based on XML

Milanovic, Gasevic et al. describe the transformation of OCL rules into Semantic Web Rule Language (SWRL) rules, using the Atlas Transformation Language (ATL) [12][19]. It should be noted, that their approach is built upon meta models for OCL and SWRL. As the focus of our paper is on the transformation UML models and OWL-2 ontologies we consider the meta models for UML and OWL-2.

Hart et al. identify three groups of features. First, features that are more or less present in both languages. Second, features that are only available in UML and third, features only offered by OWL [20]. With respect to common features, examples are used to demonstrate how these examples would appear in both languages. This collection has been incorporated in some modified form as Chapter 16 of OMG's Ontology Definition Metamodel (ODM) specification [21]. However, only part of the model elements have been considered.

Höglund et al. use MOFScript to perform a transformation from UML to OWL-2 [22]. The aim of the work is the validation of meta models. Their transformation is a model-to-text transformation. Therefore, the OWL-2 meta model is not part of the transformation.

The idea of a model transformation between UML and OWL-2 was presented by the authors in [23] und [24]. However, these publications only present the idea and cover very few selected modelling elements. Very important to us is a very careful evaluation of the transformation rules which we presented in Section VII of this article.

IX. Conclusion and Future Work

In this paper, a systematic approach for an automatic transformation of conceptual models between the Model-Driven Architecture Technology Space and the Ontology Engineering Space Technology is presented. In contrast to previous works, an approach was chosen which abstracts from the concrete syntax or XML serialization and works on the level of the meta models of UML and OWL. As a result, it was possible to show independently of individual sample models, which model elements can be transformed and which can not be transformed.

It has been found that data models written in UML can be represented as OWL Ontologies quite well. Especially when certain restrictive rules—for example those the ISO 19100 family of standards specifies— are observed, the semantics of the data model will translate well. To be mentioned as problematic are: UML's possibility to restrict the visibility of model elements, abstract classes, certain kinds of generalization (non-overlapping but not complete), aggregation and composition (which can with minor exceptions be treated as ordinary relationship types), and the extension by stereotypes.

The different extent of the meta-models clearly suggests that OWL provides much more complex means of modelling already. The transformation of general ontologies in UML data models is not always possible. Particularly problematic is the definition of element types using nested class expressions as well as sufficient conditions. But even in these cases a transformation is often possible—e.g., cardinality constraints that appear as super-types. OWL constructs such as complementation and global properties can not be transformed in general. Only under the special condition that a single element type was defined as a super-type of all other element types, a transformation is still possible.

We applied the transformation technology presented in this article to improve the quality of historic statistical data, namely the so-called "Digital Reich Statistics" (1873-1883) of the German National Library of Economics. After digitization of the original data the library established a UML model for some economic data. The transformation of this model into an OWL-2 ontology allowed us to check the consistency of the UML model and the data.

In many cases modelling concepts can be implemented in UML data models by the use of Object Constraint Language (OCL) expressions. For example, OWL constructs such as the definition of element types via sufficient conditions can be realized using OCL expressions. As a MOF-compliant abstract syntax exists for OCL, that transformation could be carried out on meta model level—like the transformations described in this article. However, the additional use of OCL results in some difficulties, such as the question of whether even all atomic OCL expressions can be represented with OWL. It might be necessary to use rule languages, e.g., the Semantic Web Rule Language (SWRL) with its built-ins. However, this would make the transformation OWL $\rightarrow$ UML more complicated. OCL is a very rich language. By nesting expressions, arbitrarily complex OCL expressions can be generated. On the one hand, the transformation of these nested expressions becomes very complex. On the other hand, it is unclear whether these complex expressions can be expressed in an OWL ontology.

In the field of comprehensibility, UML is currently superior. If there was a corresponding intuitive graphical syntax for OWL with a selection of software tools for dealing with this syntax, it would certainly contribute to increase the use of OWL in the creation of conceptual models.

REFERENCES

[1] J. Zedlitz and N. Luttenberger, "Data types in UML and OWL-2," in *SEMAPRO 2013, The Seventh International Conference on Advances in Semantic Processing*, 2013, pp. 32–35.

[2] C. Eisenhut and T. Kutzner, *Vergleichende Untersuchungen zur Modellierung und Modelltransformation in der Region Bodensee im Kontext von INSPIRE*, München, 2010.

[3] M. Horridge and S. Bechhofer, "The OWL API: A Java API for OWL Ontologies," in *Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009)*, R. Hoekstra and P. Patel-Schneider, Eds., 2009. [Online]. Available: http://ceur-ws.org/Vol-529/owled2009_submission_29.pdf

[4] A. Olivé, *Conceptual Modeling of Information Systems*, Berlin/Heidelberg/New York, 2007.

[5] M. Wahler, D. Basin, A. D. Brucker, and J. Koehler, "Efficient analysis of pattern-based constraint specifications," *Software and Systems Modeling*, vol. 9/2,, pp. S. 225–255, Heidelberg 2010. [Online]. Available: http://dx.doi.org/10.1007/s10270-009-0123-6

[6] OMG, "Unified Modeling Language, Superstructure Version 2.4," 2011. [Online]. Available: http://www.omg.org/spec/UML/2.4/Superstructure

[7] XMLSchema-2, "XML Schema Part 2: Datatypes," 2004. [Online]. Available: http://www.w3.org/TR/xmlschema-2/

[8] W. Hesse and H. Mayr, "Modellierung in der softwaretechnik: eine bestandsaufnahme," *Informatik-Spektrum*, vol. 31/5,, pp. S. 377–393, Berlin/Heidelberg 2008.

[9] H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*, Heidelberg,, 3. Auflage 2009, vol. 1.

[10] Y. Kalfoglou and M. Schorlemmer, "Ontology Mapping: The State of the Art" in *The Knowledge Engineering Review*, vol. 18/1, pp. S. 1–31, 2003. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2005/40

[11] K. Falkovych, M. Sabou, and H. Stuckenschmidt, "UML for the Semantic Web: Transformation-based Approaches," in *Knowledge Transformation for the Semantic Web*, vol. 95,, pp. S. 92–107, 2003.

[12] M. Milanović, D. Gašević, A. Guirca, G. Wagner, and V. Devedžić, "On Interchanging Between OWL/SWRL and UML/OCL," in *Proceedings of 6th Workshop on OCL for (Meta-) Models in Multiple Application Domains (OCLApps)*, 2006, pp. S. 81–95.

[13] S. Cranefield and M. Purvis, "UML as an Ontology Modelling Language," in *The Information Science Discussion Paper Series*, vol. 99/01, Dunedin 1999.

[14] S. Cranefield, "Networked Knowledge Representation and Exchange using UML and RDF," in *Journal of Digital information*, vol. 1/8, Austin 2001.

[15] K. Falkovych, "Ontology Extraction from UML Diagram,", Amsterdam, 2002.

[16] D. Gašević, D. Djuric, V. Devedzic, and V. Damjanovi, "Converting UML to OWL Ontologies," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. New York: ACM, 2004, pp. S. 488–489.

[17] D. Djurić, D. Gašević, V. Devedžić, and V. Damjanović, "A UML Profile for OWL Ontologies," in *Model Driven Architecture. European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Twente, The Netherlands, June 26-27, 2003 and Linköping, Sweden, June 10-11, 2004. Revised Selected Papers*, Berlin/Heidelberg, 2005, pp. S. 204–219. [Online]. Available: http://www.springerlink.com/content/49yb6365gymtryfg/

[18] S. Leinhos, "OWL Ontologieextraktion und -modellierung auf der Basis von UML Klassendiagrammen," Diplomarbeit, Universität der Bundeswehr München, München, 2006.

[19] M. Milanović, D. Gašević, A. Giurca, G. Wagner, and V. Devedžić, "Towards Sharing Rules Between OWL/SWRL and UML/OCL," in *Electronic Communications of the EASST Volume 5*,, 2006.

[20] L. Hart, P. Emery, B. Colomb, K. Raymond, S. Taraporewall a, D. Chang, Y. Ye, E. Kendall, and M. Dutra, "OWL Full and UML 2.0 Compared," 2004. [Online]. Available: http://www.omg.org/docs/ontology/04-03-01.pdf

[21] OMG, "Ontology Definition Metamodel," Object Management Group, 2009. [Online]. Available: http://www.omg.org/spec/ODM/1.0/

[22] S. Höglund, A. Khan, Y. Liu, and I. Porres, "Representing and Validating Metamodels using the Web Ontology Language OWL 2. TUCS Technical Report No. 973," Turku 2010. [Online]. Available: http://tucs.fi/publications/attachment.php?fname=TR973.full.pdf

[23] J. Zedlitz, J. Jörke, and N. Luttenberger, "From UML to OWL 2," in *Proceedings of Knowledge Technology Week 2011*, D. Lukose, A. R. Ahmad, and A. Suliman, Eds., Berlin/Heidelberg, 2012, pp. p. 154–163.

[24] J. Zedlitz and N. Luttenberger, "Transforming Between UML Conceptual Models and OWL 2 Ontologies," in *Proceedings of the Terra Cognita Workshop on Foundations, Technologies and Applications of the Geospatial Web, in conjunction with the 11th International Semantic Web Conference (ISWC 2012)*, D. Kolas, M. Perry, R. Grütter, and M. Koubarakis, Eds., 2012, pp. p. 15–26. [Online]. Available: http://ceur-ws.org/Vol-901/paper2.pdf

A Technique to Avoid Atomic Operations on Large Shared Memory Parallel Systems

Rudolf Berrendorf
Computer Science Department
Bonn-Rhein-Sieg University
Sankt Augustin, Germany
e-mail: rudolf.berrendorf@h-brs.de

***Abstract*—Updating a shared data structure in a parallel program is usually done with some sort of high-level synchronization operation to ensure correctness and consistency. The realization of such high-level synchronization operations is done with appropriate low-level atomic synchronization instructions that the target processor architecture provides. These instructions are costly and often limited in their scalability on larger multi-core / multi-processor systems. In this paper, a technique is discussed that replaces atomic updates of a shared data structure with ordinary and cheaper read/write operations. The necessary conditions are specified that must be fulfilled to ensure overall correctness of the program despite missing synchronization. The advantage of this technique is the reduction of access costs as well as more scalability due to elided atomic operations. But on the other side, possibly more work has to be done caused by missing synchronization. Therefore, additional work is traded against costly atomic operations. A practical application is shown with level-synchronous parallel Breadth-First Search on an undirected graph where two vertex frontiers are accessed in parallel. This application scenario is also used for an evaluation of the technique. Tests were done on four different large parallel systems with up to 64-way parallelism. It will be shown that for the graph application examined the amount of additional work caused by missing synchronization is neglectible and the performance is almost always better than the approach with atomic operations.**

***Index Terms*—atomic operation, CAS, scalability, shared memory, redundant work, parallel work queue, parallel Breadth-First Search**

I. Introduction

Updating a shared data structure in a parallel program as for example the state check/update whether a vertex in a graph is visited or not [1] is usually done on an application level with some sort of high-level atomic update operation. In OpenMP [2] this could be realized lock-protected, in a critical section, or if syntax allows with an atomic pragma construct. Pthread-related API's (Application Programming Interface) [3] [4] [5] can use for example a mutex variable to protect access to a shared data structure and to ensure mutual exclusion of parallel threads. A general discussion on using different synchronization constructs in parallel and concurrent programming can be found in [6] and [7].

The implementation of such a high-level synchronization operation itself is done by the compiler or inside a runtime system often with one or even more atomic instructions (Compare-And-Swap, Atomic-Add, Fetch-And$-\Phi$, Test-And-Set, etc.) of the underlying processor architecture [8] [9] [10]. The general problem with atomic instructions of type read-modify-write is that these are rather costly compared to ordinary memory accesses and not really scalable on larger systems [11] [12] (see also Section IV for investigations on that). The time for *one* such atomic operation increases significantly under contention as the number of cores in a multi-core / multi-processor system gets larger. Therefore, frequently accessing shared data structures with atomic operations imposes a severe performance problem, especially on large parallel systems.

The use of such synchronized updates on shared data guarantees correct operations on the data when using multiple threads. But this strict enforcement is often not really necessary. An example is a work queue, where working threads insert new work items and idle threads remove such items to be worked on. But for certain algorithmic scenarios (e.g., within a certain program phase), a work item may be inserted even multiple times without violating the *overall* correctness of the algorithm, but only causing additional redundant work to be done as the same work item may exist multiple times in a queue. In such cases, the costly synchronized access could be completely removed and replaced with cheaper non-atomic accesses, but eventually introducing additional work to be done if work items get inserted multiple times.

An example for such a scenario is a Breadth First Search (BFS) for undirected graphs (see Section III for details). Many of the published parallel BFS algorithms iterate over a vertex frontier where the vertices of the current vertex frontier determine, which unvisited vertices are part of the following vertex frontier. In this scenario, adding a vertex twice in such a frontier generates more work to be done in the next level iteration but does not influence the correctness of the algorithm (see Section V for details). Another, more general scenario is the development of asynchronous algorithms [13] [14].

In this paper, a general optimization strategy is introduced that replaces costly atomic modifiers with cheaper read/write accesses. Necessary conditions are defined that need to be met to apply the technique. The motivation for this optimization technique and the evaluation is done using a concrete parallel BFS algorithm on large shared memory multi-core multi-processor systems with up to 64 cores. Factors are discussed that influence the amount of potential additional work and whether this additional work without any synchronized access trades off against the traditional synchronized access to a work

queue doing exactly the amount of work that is necessary.

The paper is organized as follows. After the introduction, the paper starts with an overview of related work. After this, a brief overview is given on parallel BFS algorithms; level synchronous BFS is an example where the new approach can be applied and will also be used in the evaluation of the new technique. In Section IV, evidence is given that certain atomic operations including Compare-And-Swap have scalability problems on larger systems. In Section V, the new approach is introduced first for a special scenario, a generalization of the technique follows in Section VI. After that, the experimental setup is pointed out, and then the new approach is evaluated in detail. The paper ends with a summary.

II. Related Work

There are several papers on certain aspects on the optimization of synchronization constructs in a wider sense. This includes, amongst others, reducing the number of consecutive mutex locks/unlocks [15] in a program and compiler optimizations for read/write barriers [16]. Furthermore, there are advanced synchronization techniques trying to minimize synchronization costs including RCU (Read-Copy-Update) [17], special monitors [18] and read-writer optimizations [19].

An interesting general approach to handle possible concurrent accesses to shared data structures is the concept of transactional memory (original concept paper [20]). This approach has some similarities with the approach introduced in this paper as both are optimistic: do a read-modify-write operation without a critical section and react only is something went wrong. The idea with transactional memory as well as in the new approach discussed later in this paper is that the bad thing happens rather seldom. Transactional memory detects the problem and, depending on the Application Program Interface (API) in use, rolls back the whole transaction and restarts the operation. The technique proposed in this paper instead ignores the problem (and does not even detect the problem) and has more work to do in the remaining execution of the algorithm. Transactional memory is implemented on a hardware level in recent processors (IBM processors PowerPC for BlueGene/Q [21] and System z [22]; Intel Haswell [23]).

Lock-free and wait-free data structures are often proposed as a way to reduce/avoid synchronization problems (priority inversion, deadlock) that may occur using mutual exclusion or other blocking synchronization constructs. Starting with the fundamental idea by Leslie Lamport [24], many papers followed on certain aspects, e.g., [25] [26] [27] [28] [29] [30] [31] [32] [33] [34]; see [35] for a comprehensive view. As lock- and wait-free data structures are internally often realized with (as will be shown, non-scalable) atomic Compare-And-Swap operations, similar ideas as presented in this paper might be interesting in that area, too.

In [36], a parallel graph algorithm for the construction of a spanning tree is discussed using mutual exclusion and lock-free data structures. The authors discuss the problem of overlapping work and how to ensure that every work item (newly visited vertex of the graph) is handled by one thread only using atomic Test-And-Set operations. In our paper, it is proposed the other way to allow that a work item may be handled by more than one thread and avoiding even a test-and-set operation that is still necessary in [36].

Reference [6] gives an overview of different aspects on related topics. [37] shows a similar benign race as ours in a parallel BFS algorithm, but without analyzing the influence of that in detail.

III. Parallel Algorithms for BFS

Parallel level-synchronous BFS algorithms will be used as a motivating application as well as in the evaluation of the new idea that is introduced later in detail. Therefore, a short introduction to parallel BFS follows. Breadth-First Search is a visiting strategy for all vertices of a graph. BFS is most often used as a building block for many other graph algorithms, including single-source shortest paths, minimum spanning tree, connected components, bipartite graphs, maximum flow, and others [38] [39]. Additionally, BFS is used in many application areas where certain application aspects are modeled by a graph that needs to be traversed according to the BFS visiting pattern. Amongst others, exploring state space in model checking, image processing, investigations of social and semantic graphs, machine learning are such application areas [40].

In the application scenario used for the examination, undirected graphs $G = (V,E)$ are of interest, where $V = \{v_1, ..., v_n\}$ is a set of vertices and $E = \{e_1, ..., e_m\}$ is a set of edges. An edge e is given by an unordered pair $e = (v_i, v_j)$ with $v_i, v_j \in V$. The number of vertices of a graph will be denoted by $|V| = n$ and the number of edges is $|E| = m$.

Assume a connected graph and a source vertex $v_0 \in V$. For each vertex $u \in V$ define $depth(u)$ as the number of edges on the shortest path from v_0 to u, i.e., the edge distance from v_0. With $depth(G, v_0)$ the depth of a graph G is denoted defined as the maximum depth of any vertex in the graph *relative to the given source vertex* v_0. Please be aware, that this may be different to the diameter of a graph, the largest distance between *any* two vertices.

The problem of Breadth First Search for a given graph $G = (V,E)$ and a source vertex $v_0 \in V$ is to visit each vertex in a way such that a vertex v_1 must be visited before any vertex v_2 with $depth(v_1) < depth(v_2)$. As a result of a BFS traversal, either the level of each vertex is determined or a (non-unique) BFS spanning tree with a father-linkage of each vertex is created. Both variants can be handled by BFS algorithms with small modifications and without extra computational effort. The problem can be easily extended and handled with directed or unconnected graphs. A sequential solution to the problem can be found in textbooks, based on a queue where all non-visited adjacent vertices of a visited vertex are enqueued [38] [39]. The computational complexity is $O(|V + |E|)$.

If one tries to design a parallel BFS algorithm, different challenges might be encountered. As the computational density of BFS is rather low, BFS is bandwidth limited for large graphs and therefore memory bandwidth has to be handled

with care. For a similar reason in cache coherent NUMA systems (Non-Uniform Memory Access [41]), data layout and memory access should respect processor locality. In multicore multiprocessor systems, things get even more complicated, as several cores share higher level caches and NUMA-node memory, but have private lower-level caches.

```
1:  function BFS(graph g, vertex source)
2:    var
3:      d, distance vector of size |V|. Initial values: ∞
4:      current,next, vertex container. Initially empty
5:    end var
6:    d[source] ← 0
7:    current.insert(source)
8:    while current is not empty do
9:      for all v in current do
10:       for all neighbours w of v do
11:         old = CompareAndSwap(d[w],∞,d[v]+1)
12:         if old == ∞ then
13:           next.insert(w)
14:         end if
15:       end for
16:     end for
17:     Barrier
18:     swap current with next
19:   end while
20:   return d
21: end function
```

Fig. 1: Parallel BFS with an atomic Compare-And-Swap-operation

In BFS algorithms housekeeping has to be done on visited / unvisited vertices with several possibilities how to do that. Some of them are based on special container structures for vertex frontiers where information has to be inserted and deleted. Scalability and administrative overhead of these containers are of interest. Generally speaking, these approaches deploy two identical containers (current frontier, next frontier) whose roles are swapped at the end of each level iteration. Fig. 1 shows this in a rather straightforward formulation with an atomic Compare-And-Swap (CAS) operation in an inner loop (line 11) to detect and update unvisited vertex neighbors. In this atomic operation, a vertex w is checked whether it is visited already ($d[w] \neq \infty$), and if not, marks the vertex as visited. Based on this knowledge, only an unvisited vertex gets inserted into the next vertex frontier. After all vertices in the current container are visited, all threads wait at a barrier before work on the next container / frontier gets started (level iteration). This version can be further optimized using chunked lists for every thread. The insert operation of a new vertex into a thread-local chunk can be done in a non-atomic way. But the construction of a global list from thread-local chunks (i.e., the insertion of each chunk into a global list) must still be done in a synchronized way. But as this is done only if a chunk gets full, this is not the critical operation of this algorithm but the detection of whether a vertex is visited or not in line 11. Container centric approaches are eligible for dynamic load balancing but are sensible to data locality on NUMA systems. Container centric approaches for BFS can be found in some parallel graph libraries [42] [43]. Reference [44] contains an overview and evaluation of several parallel BFS algorithms.

For level synchronized approaches, a simple list is a sufficient container. There are approaches in which each thread manages two private lists to store the vertex frontiers and uses additional lists as buffers for communication [45] [46]. This approach deploys a static one dimensional partitioning of the graph's vertices and therefore supports data locality.

Level synchronous algorithms are quite easy to understand and often to realize, too. With certain additional optimizations performance is often very good [44]. This type of BFS algorithm is used as an instance of the problem scenario where the newly proposed technique can be applied. It should be mentioned, that for certain classes of graphs (e.g., high diameter graphs) parallel algorithms exist that perform better than the level synchronous approach [47] [48] [49] [50].

TABLE I: PROCESSORS AND SYSTEMS USED

name	Intel-IB	Intel-SB	AMD-IL	AMD-MC
processor:				
manufacturer	Intel	Intel	AMD	AMD
CPU name	E5-2697	E5-2670	Opteron 6272	Opteron 6168
architecture	Ivy Bridge	Sandy Bridge	Interlagos	Magny Cours
frequ.[GHz]	2.7	2.6	2.1	1.9
system:				
memory [GB]	256	128	128	32
# CPU sockets	2	2	4	4
n-way parallel	48	32	64	48

IV. Performance Problem of Atomic Read-Modify-Write Operations on Large Parallel Systems

Atomic operations in a higher level parallel API for shared memory systems as mutual exclusion, atomic update, locks, Compare-And-Swap etc. [6] [7] [51] [52] are usually mapped on shared memory systems to atomic instructions that the underlying processor architecture provides [8] [9] [10]. Some of these atomic instructions are by itself rather costly compared to a simple memory access if no contention exists. For exampe, embedded in a function call and executed by one thread on an Intel E5-2697 CPU, an atomic CAS operation on a 64 bit data type takes 7 ns compared to 3 ns that a compound non-atomic read-test-write operation takes. Table I describes in detail and names the systems used in the following.

But under contention, if multiple threads concurrently access a shared state with such instructions, the cost *per operation* increases significantly for certain types of atomic operations. This is especially true for read-test/modify-write type operations like Compare-And-Swap and Fetch-And-Add. And the contention penalty gets higher the more CPU sockets a system has [11] [12]. Fig. 2 shows the cost for one 64-bit atomic Compare-And-Swap operation (of type read-modify/test-write) on different shared memory systems dependend on the number of threads utilised. The number of threads used does not exceed the degree of hardware parallelism a system under

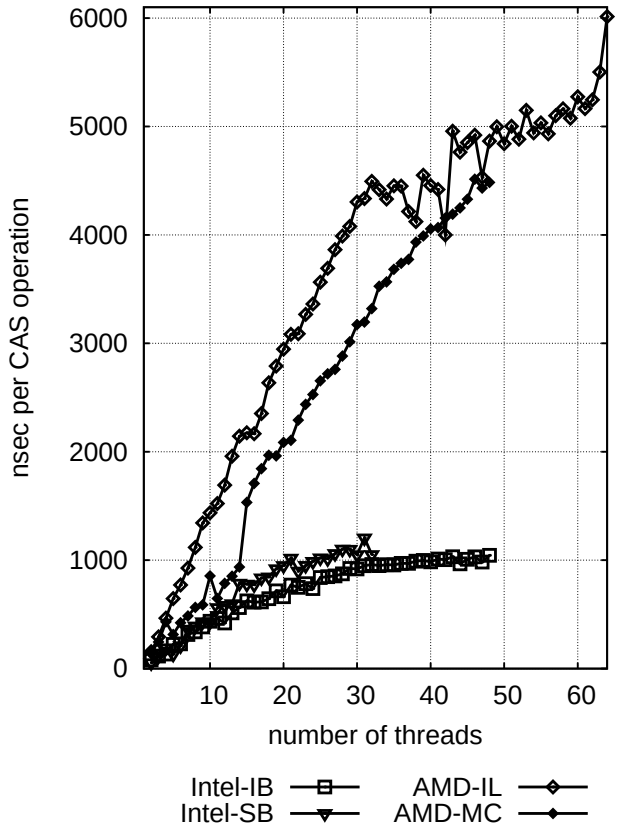


Fig. 2: Cost per Compare-And-Swap operation on different parallel systems.

consideration has. Therefore, every thread is mapped (in an operating system specific manner) to a different core of the system and is always runnable (ignoring operating system effects like interrupt handling). In this test, p threads do in parallel in a loop $n = 1,000,000$ atomic CAS operations each. The test was executed on parallel systems of different generations of processors, processor manufacturers, degree of parallelism, and number of processor sockets. Similar results can be seen for other atomic read-test/modify-write operations, too, e.g., atomic Fetch-And-Add. As can be easily seen from the results, the overhead on larger systems with four sockets (the AMD-based systems used) is significantly higher than on smaller systems with two sockets (the Intel-based systems used). The minimum time for a CAS operation is between 7 and 22 ns without contention on the systems used, the time gets as high as approx. 1,000 ns per operation on the two socket systems and up to approx. 6,000 ns on the four socket systems under heavy contention. This means that performance problems with atomic operations hurt on larger systems with more sockets even more than on smaller systems.

V. Alternative to Atomic Accesses

In this section, a technique is proposed to replace the costly and non-scalable atomic accesses with cheaper non-atomic accesses. First, the technique will be motivated on the concrete BFS application introduced in Fig. 1. It will be shown, that under certain conditions atomic operations can be traded against additional work and therefore traded against additional overhead. Then a discussion follows, what factors influence this possible overhead. And finally, in Section VI, a generalization is given under what circumstances the technique can be applied generally.

A. Optimizing BFS

Looking at the formulation of the parallel BFS algorithm in Fig.1, an atomic CAS-Operation is used in line 11 to check whether the neighbour vertex w is unvisited ($d[w] = \infty$), and if so, replace the depth-value of w with the depth value of the current vertex v incremented by one. And if the neighbour vertex w was unvisited, additionally insert w into the next vertex frontier. The replacement of the value ∞ by a non-∞ value (the depth value) marks the vertex as visited. The CAS operation guarantees, that every vertex is inserted exactly once into a vertex frontier (detection and mark of visitedness). Without the atomic operation, a race condition [53] exists on $d[w]$. Replacing the critical operation with a non-atomic code results in Fig. 3 (only relevant parts are shown here).

```
1: for all neighbours w of v do
2:   if d[w] = ∞ then
3:     d[w] = d[v] + 1
4:     next.insert(w)
5:   end if
6: end for
```

Fig. 3: Parallel non-atomic BFS (relevant part)

The code of interest is in lines 2 and 3, that was previously guarded by the CAS-operation. There are two possibilities when executing this code in parallel:

1) Between the read access $d[w]$ in line 2 and the completion of the write access in line 3 no other thread accesses $d[w]$. In this case, there is *no problem* with this version, the vertex w is inserted exactly once in a vertex frontier as before. But see additionally the discussion of the appropriate memory model below.
2) More than one thread detects for a certain vertex w_x that w_x is unvisited (i.e., $d[w_x] = \infty$) before any of the other threads can change the $d[w_x]$ to some visited value. In this case, the vertex w_x gets inserted twice or even more into the next vertex frontier.

The insert operation in line 4 has to be done with care as this might be done concurrently by multiple threads. As this has to be handled in all version similar and is not really critical in all versions of discussion, this is not further discussed here. It is important to state that even the second case produces *no wrong results* as *any thread* that detects that $d[w_x]$ is unvisited, writes into $d[w_x]$ in the next step the value $d[v]+1$ that is *the same value for all threads* in one level iteration. Therefore, correctness is guaranteed in our scenario even if multiple threads concurrently detect that the same vertex is unvisited. But, as stated above, in such a case the vertex w_x is inserted twice or even more into the next vertex frontier and due to

that, generates more and redundant work in the next level iteration. Working later on a vertex multiple times is again no correctness problem. The resulting depth values for all vertices are the same in a level-synchronous algorithm, independend on how many times a vertex gets worked on.

Inspecting the generated assembler code for lines 2 and 3 of the code given in Fig. 3, the read access to $d[w]$ in line 2 (i.e., a load instruction) and the write access to $d[w]$ in line 3 (i.e., a store instruction) are nearby instructions in the code sequence. These inspections were done for different compilers (GNU gcc, Intel icc, PGI pgcc) and it was found that this observation is more or less invariant of the compiler used for the given code sequence (and it would be curious if this observation would not hold for this code example). With an assumption, that a thread is not suspended during execution, the time window between the two instructions is therefore rather small (few cycles in practice). This assumption will be mostly true for many real scenarios where parallel programs get executed, e.g., running OpenMP programs on a dedicated system with not more threads than processor cores available.

Another aspect in this discussion is the memory consistency model in use [54] [55] [56] [57]. In a strict memory consistency model, it is guaranteed, that a read operation always retuns the value of the last write operation to that memory location. But todays, all memory consistency models in practical use (e.g., [5] [4] [2] [58]) are rather relaxed and the compiler may buffer the value of $d[w]$ in a register, a processor core may buffer that value in write buffers, or the new value is not propagated between different processors soon, etc. This can enlarge the time window for problems substantially even under the assumption made above that a thread is not suspended. A programmer may insert an appropriate flush operation before line 2 and after line 3 such that all threads / processors are forced to read / write $d[w]$ to / from main memory in the corresponding operation. But dependend on the implementation of such a flush-operation, this could lead to substantial additional overhead as this is for example done inside an inner loop iteration in the application example given.

B. Factors Influencing Additional Work

The question of interest is now, whether the relaxation using non-atomic modifications to $d[w]$ as given in Fig. 3 (which surely is faster than a CAS-operation as explained in Section IV) pays off. Due to the fact that without an atomic CAS operation a vertex might get inserted multiple times into the next vertex frontier, the question is whether the amount of work to be done might be increased substantially. The amount of additional work to be done will be influenced generally speaking by:

1) the problem time window in relation to the time threads spend in non-critical code. This is influenced by the generated code sequence and implemented consistency model as discussed above.
2) the number of threads in use, i.e., the number of concurrent parties.
3) the problem data influencing access collisions, i.e., in our case the topology of the graph (vertex degrees, shared neighbours)

The larger the time window is that another thread may see the vertex in question as unvisited, and the more threads are participating, and the more vertices have connections to the unvisited vertex, the higher the probability that additional work is generated.

VI. Generalization of the Technique

Although the motivation of the technique was given here in the context of a parallel BFS algorithm, the technique itself is not specific to BFS and can be generalized. Therefore, the suggestion is to replace costly atomic operations with cheaper simple load/store operations without influencing the correctness of the algorithm but probably doing more / redundant work. The hypothesis is that especially on large shared memory systems with many concurrent threads this technique pays off.

Looking in a more abstract way on the suggested technique, there is predicate $p : X \to \{true, false\}$ for some set X. If the result of the predicate is true, there is a state-change operation $c : X \to X$ for the same $x \in X$ that the predicate was applied to, followed by some operation $f : Y \to Y$ for some set Y. The generalized application scenario is therefore given in Fig. 4.

```
1: if p(x) then
2:     c(x)
3:     f(y)
4: end if
```

Fig. 4: Generalized Scenario

As multiple threads might execute the predicate p, multiple threads might detect the same true condition and therefore execute $c(x)$ and $f(y)$ subsequently and also redundantly. As a consequence, the operation c and f must be both idempotent to ensure that executing the state-change function f multiple times does not influence the correctness. A function $g : A \to A$ is called *idempotent* if $g \circ g = g$, i.e., $\forall a \in A : g(g(a)) = g(a)$.

For the BFS example, the p-Operation is the test $d[w] = \infty$ and the c-operation changes the state of $d[w]$ to the same value if executed multiple times for the same vertex. The f-operation is the insertion of the vertex into the next vertex front. The c-operation as well as the f-operation meet the requirements for the two operations, respectively. As $d[w]$ is assigned the same value, this is an idemponent operation. And the multiple insertion of a vertex into the next vertex frontier is also (semantically) a idemtpotent operation, because the *result* of working on the next vertex frontier will be the same, independent how many times a vertex gets inserted into a frontier.

VII. Experimental Setup

In this section, the technique introduced in Section V is systematically examined with the concrete scenario of parallel BFS. Three factors were identified that may influence the

performance of an application using the new technique due to additionally generated work. All factors are examined in detail.

A. Algorithm Versions

The general algorithmic approach for parallel BFS chosen for this discussion was already given in Fig. 1 in an easy to formulate version. The concrete realization to handle vertex fronts was done for this evaluation with chunked array based lists where each thread inserts a new vertex unprotected into a thread-private chunk of size 128. If such a chunk gets filled, the chunk is inserted into a global list in a protected way. The insertion of a whole chunk into the global list is done in all algorithm versions examined with one atomic operation. But the influence of that atomic operation is rather small as only whole chunks get inserted and not single vertices.

In the first version named *atomicBFS*1 (see Fig. 5), every thread uses a CAS operation as described in Fig. 1 to detect unvisited vertices and updates them accordingly. This guarantees, that every vertex is inserted exactly once in a vertex frontier. But on the other side, *every* check is done atomically even on vertices that were visited already, even in any previous level iteration. This is a save but somewhat naïve implementation.

This last aspect can be optimized for many program kernels easily with a standard optimization technique for parallel programs in prefixing the expensive CAS-operation with a normal read operation followed by the CAS-operation *only*, if the test was successful, i.e., a test-and-test-and-set operation (see Fig. 6). This technique has significant advantages if vertices get visited many times (e.g., graphs with high vertex degrees). Then, only the first visit must be atomic, all other accesses would detect that the vertex is visited already. This optimization technique is also used in the OpenMP reference implementation of the Graph500 benchmark [42] for BFS. This optimized version is named *atomicBFS*2. In this version, all vertices already visited are no longer handled with a CAS operation. The discussion of the performance effect of this optimization is given later in detail.

The third approach named *nonatomicBFS* does not use atomic operations for the detection of unvisitedness, but rather uses the technique proposed (see Fig. 7). Therefore, a vertex may be inserted more than once in the next vertex frontier. The main difference to the other two versions is therefore that the detection of an unvisited vertex and the subsequent update to a visited state is no longer done atomically but rather with simple read/write accesses including the possibility of multiple insertions of a vertex as multiple threads may see concurrently a vertex as unvisited. Further algorithmic optimizations different to that discussed here and a general overview of parallel BFS algorithms can be found in another paper [44]. There is also shown, that there are better but more complex algorithms for the parallel BFS problem for large shared memory systems. In this paper, only the discussion atomic operations vs. redundant work is of interest, and therefore the *relative* comparism of the introduced three versions is sufficient for that.

To filter out unrelated effects, all test runs were repeated 5 times and the best result out of these 5 results was taken as the final result of a test.

```
1: for all neighbours w of v do
2:    old = CompareAndSwap(d[w], ∞, d[v] + 1)
3:    if old == ∞ then
4:       next.insert(w)
5:    end if
6: end for
```

Fig. 5: Kernel for *atomicBFS*1

```
1: for all neighbours w of v do
2:    if d[w] ≠ ∞ then
3:       old = CompareAndSwap(d[w], ∞, d[v] + 1)
4:       if old == ∞ then
5:          next.insert(w)
6:       end if
7:    end if
8: end for
```

Fig. 6: Kernel for *atomicBFS*2

```
1: for all neighbours w of v do
2:    if d[w] == ∞ then
3:       d[w] = d[v] + 1
4:       next.insert(w)
5:    end if
6: end for
```

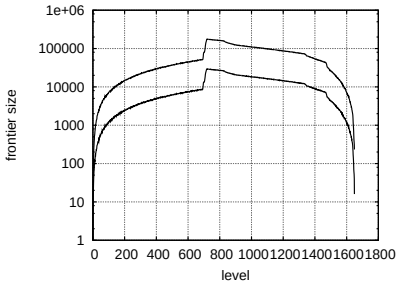
Fig. 7: Kernel for *nonatomicBFS*

B. Factors Influencing Overhead

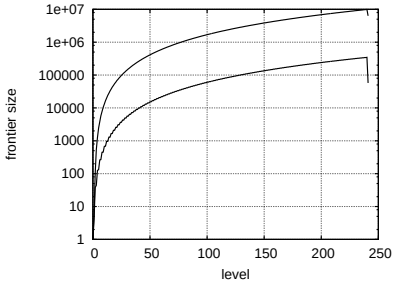
As discussed already in Section V, the first factor influencing the probability of multiple insertions is the time window related to the time spent in non-critical code. Although the BFS algorithm has only few instructions between the read and write operation on the critical data, there is not much work to do in the non-critical part (just the insert operation) executing the critical part with high frequence and therefore increasing the probability for collisions. Therefore, BFS is an example for a rather problematic algorithm in this sense.

The second factor influencing the probability of double insertion is the degree of parallelism. Different parallel systems were used in the tests as described already in Table I. The largest one is a 64-way AMD-6272 Interlagos based system with 128 GB shared memory organised in 4 NUMA nodes (i.e., 4 CPU sockets). A second AMD-based systems has also four sockets, while the remaining two systems are two-socket systems (see Table I for details and names to refer to a specific system).

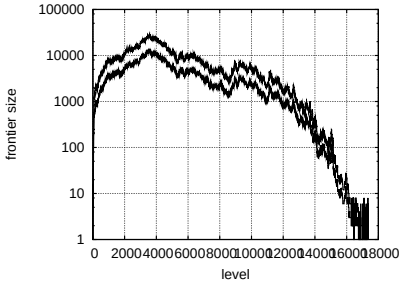
The third factor influencing additional work is the probability of a data collision, i.e., in the given application the probability that two vertices with the same depth share a common unvisited neighbor in the graph. Only unvisited neighbours leed to an atomic operation in version *atomicBFS*2 and to



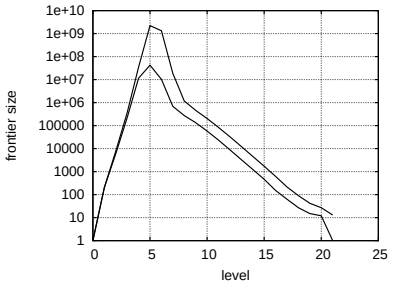
(a) Frontiers for delaunay



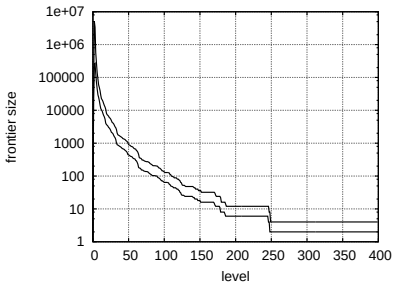
(b) Frontiers for nlpkkt240



(c) Frontiers for road-europe

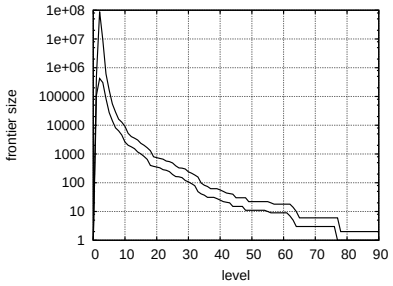


(d) Frontiers for friendster

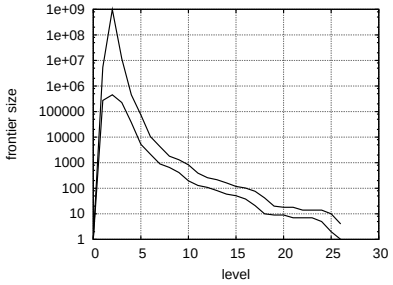


(e) Frontiers for R-1M-10M-57

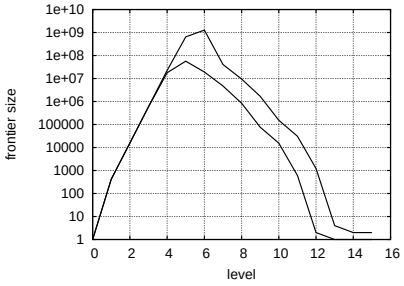
Fig. 8: Vertex and edge frontier sizes of selected graphs (part 1). Upper curve is edge frontier, lower curve is vertex frontier.



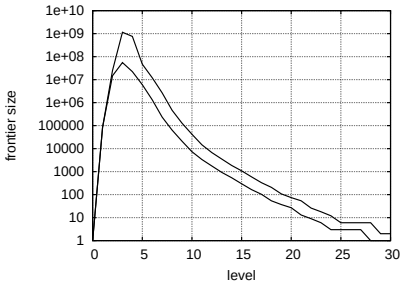
(a) Frontiers for R-1M-100M-57



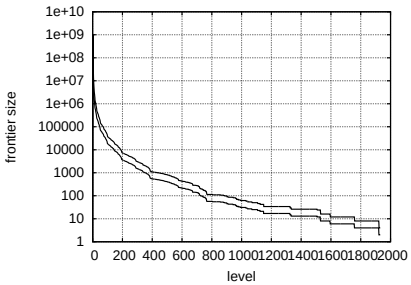
(b) Frontiers for R-1M-1G-57



(c) Frontiers for R-100M-2G-30



(d) Frontiers for R-100M-2G-45



(e) Frontiers for R-100M-2G-57

Fig. 9: Vertex and edge frontier sizes for selected graphs(part 2). Upper curve is edge frontier, lower curve is vertex frontier.

TABLE II: CHARACTERISTICS FOR USED GRAPHS

graph name	$\|V\| \times 10^6$	$\|E\| \times 10^6$	degree avg.	degree max.	graph depth
delaunay [59]	16.7	100.6	6	26	1650
nlpkkt240 [60]	27.9	802.4	28.6	29	242
road-europe [59]	50.9	108.1	2.1	13	17345
friendster [61]	65.6	3612	55	5214	22
R-1M-10M-30	1	10	10	107	11
R-1M-10M-45	1	10	10	4726	16
R-1M-10M-57	1	10	10	43178	400
R-1M-100M-30	1	100	100	1390	9
R-1M-100M-45	1	100	100	58797	8
R-1M-100M-57	1	100	100	530504	91
R-1M-1G-30	1	1000	1000	13959	8
R-1M-1G-45	1	1000	1000	599399	8
R-1M-1G-57	1	1000	1000	5406970	27
R-100M-1G-30	100	1000	10	181	19
R-100M-1G-45	100	1000	10	37935	41
R-100M-1G-57	100	1000	10	636217	3328
R-100M-2G-30	100	2000	20	418	16
R-100M-2G-45	100	2000	20	85494	31
R-100M-2G-57	100	2000	20	1431295	1932
R-100M-4G-30	100	4000	40	894	15
R-100M-4G-45	100	4000	40	180694	31
R-100M-4G-57	100	4000	40	3024348	1506

a possible double-insertion in version *nonatomicBFS*. This factor is mainly influenced in the given scenario by the graph topology / degree distribution. To examine this influence, several large graphs were used from different application domains including real graphs from social networks, road networks, optimization problems, and triangulation graphs. The graph instances were taken from then DIMACS-10 challenge [59], the Florida Sparse Matrix Collection [60], and the Stanford Large Dataset Collection [61]. The graph `friendster` and larger RMAT-graphs could not be used on all systems due to memory requirements. Additionally, synthetically generated pseudo-random graphs were used that guarantee certain topological properties. R-MAT [62] is such a graph generator with parameters a,b,c influencing the topology and clustering properties of the generated graph (see [62] for details). R-MAT graphs are mostly used to model scale-free graphs. For the evaluation tests, graphs of the following classes werde used:

- Graphs with a very low average and maximum vertex degree resulting in a rather high graph depth and limited vertex fronts. A representative for this class is the road network `road-europe`.
- Graphs with a moderate average and maximum vertex degree. For this class, Delaunay graphs representing Delaunay triangulations of random points (`delaunay`) and a graph for a 3D PDE-constraint optimization problem (`nlpkkt240`) are used.
- Graphs with a large variation of degrees including few very large vertex degrees. Related to the graph size, they have a smaller graph depth. For this class of graphs, a real social network (`friendster`), and synthetically generated Kronecker R-MAT graphs are used, the later with different vertex and edge counts and three R-MAT parameter sets. The first parameter set named 30 is $a = 0.3, b = 0.25, c = 0.25$, the second parameter set 45 is $a = 0.45, b = 0.25, c = 0.15$, and the third parameter set named 57 is $a = 0.57, b = 0.19, c = 0.19$.

All test graphs are connected, for R-MAT graphs guaranteed with $n-1$ artificial edges connecting vertex i with vertex $i+1$. Some important graph properties for the graphs used are given in Table II. For a general discussion on degree distributions of R-MAT graphs see [63].

C. Factors Influencing Performance and Scalabilty

To interpret results in the following section, frontier sizes during the level-synchronous execution of the BFS algorithm will be given in relation to the level number as an additional information (see Figs. 8 and 9). The *edge frontier size* gives the number of outgoing edges from vertices in the current frontier, i.e., the number of vertex *candidates* that have to be checked for inclusion into the next frontier. On the other side, the *vertex frontier size* gives the number of unique vertices that get inserted into the next vertex frontier, i.e., the vertex was checked, found unvisited, and then sucessfully inserted. The edge frontier size is therefore the amount of checks to be done (in algorithm version *atomicBFS*1 with a CAS operation, in the other versions by a simple read operation), and the vertex frontier size is the amount of actual insertions into the next frontier (in version *atomicBFS*2 as part of the CAS, in version *nonatomicBFS* with a simple write). The edge frontier size is always at least as large as the vertex frontier size. In the figures, the edge frontier size is always the upper curve.

Setting this frontier information in relation to the performance numbers, a large difference between edge frontier size and vertex frontier size in a level iteration means that many atomic checks were made in version *atomicBFS*1 that did not lead to an unvisited neighbor vertex / insert operation. On the other side, if the difference between vertex and edge frontier size is small, the difference between the two atomic algorithm versions should be less as most of the atomic operations are executed in both atomic versions.

Figs. 8 and 9 show frontier sizes during each level. Please be aware that the y-axis has a logarithmic scale. The higher a number for the vertex frontier is, the more parallel work is available. A frontier size of 1,000 or even less on a parallel system with 64 threads all working in parallel on this problem means a severe performance limitation.

All BFS algorithms introduced here are limited for large graphs by memory bandwidth demands, especially when using many threads. This means that for many large graphs and using many threads, the effects under discussion here may be hidden by memory bandwidth restrictions [44]. Additionally, if there is not enough parallelism available (small vertex frontier at any level iteration), performance is again limited in all versions using many threads. In such situations, algorithms *atomicBFS*2 and *nonatomicBFS* will most likely perform very similar. In Section VIII-C, statistical filters are used to handle this in the discussion.

VIII. Evaluation Results

In the following, performance results are discussed comparing the three algorithm versions on the different parallel systems and with different input data as specified in Section VII. Not all results can be shown here in detail. Rather, the influence of the stated factors is discussed, results are summarized where reasonable, and overall statistics are presented. Additionally, the amount of overhead for *nonatomicBFS* is discussed. Performance number for BFS will be given as a rate Million Traversed Edges per Second (MTEPS), a usual measure for BFS performance [42] (the higher, the better).

A. Absolute Performance Results

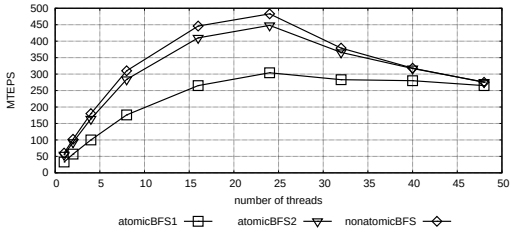
Figs. 10 - 13 show absolute performance results for the three algorithm versions of investigation on the different parallel systems using selected data sets. The performance limitation or even degradation using many threads (especially with graph `road-europe`) is caused by the limited parallelism (see Figs. 8a and 8c for that) or memory bandwidth restrictions. More sophisticated BFS algorithms that are out of the scope of this paper can handle that more efficient. Details on that can be found in [44].

For all graphs shown other than `road-europe` the algorithm version performing worst is the algorithm version *atomicBFS*1 as with *every* access to $d[w]$ in the relevant code section an atomic CAS operation is executed. This is true on all systems used and with nearly all thread counts. The performance difference to the other two algorithm versions is very high, if many of the atomic operation were done unneccesarily, i.e., a vertex of investigation was visited already before. This is the case if the difference between edge and vertex frontier is large. And the performance difference is large as long as no other effects (limited parallelism, memory bandwidth restriction) superimpose this effect. Both other algorithm versions use no CAS at all (*nonatomicBFS*) or only if a vertex has been seen in a non-atomic pre-test as unvisited (*atomicBFS*2). This behaviour clearly underpins the central message that atomic operations should be avoided whenever possible.

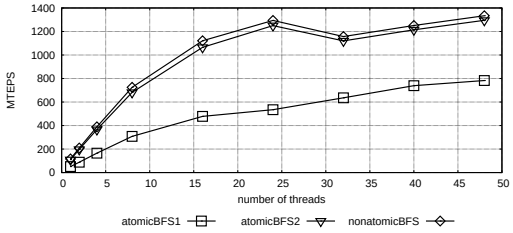
B. Relative Improvements

As algorithm version *atomicBFS*1 has in most configurations severe performance limitations, a closer look will be done on the other two version only: *atomicBFS*2 using CAS only if necessary and *nonatomicBFS* using the introduced technique of avoiding atomic operations at all.

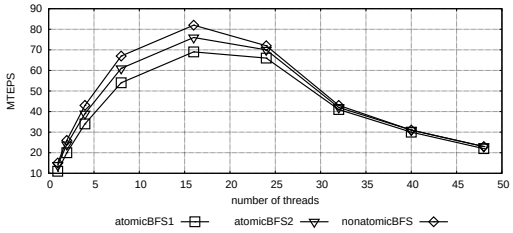
Fig. 14 shows relative performance improvements in percent of algorithm version *nonatomicBFS* without atomic operations relative to algorithm version *atomicBFS*2 with (already optimized) atomic updates. A positive value means that the non-atomic version performs better than *atomicBFS*2. The figures show that for many threads the difference between the two versions of discussion is rather small (and then often below the accuracy of measurement). The reason for that was given already: with this rather simple BFS algorithm versions, for many threads and large graphs, memory bandwidth and/or limited parallelism is the limiting factor, and not the atomic



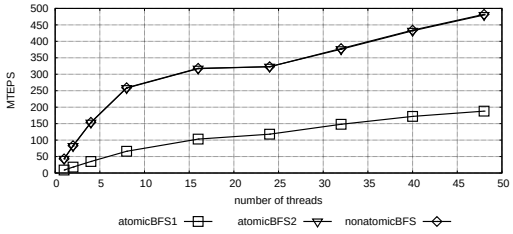
(a) delaunay



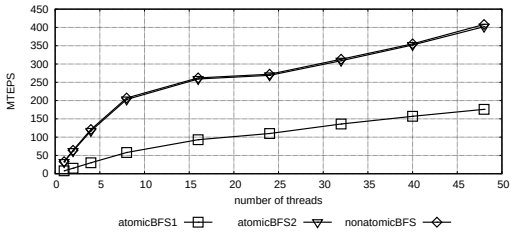
(b) nlpkkt240



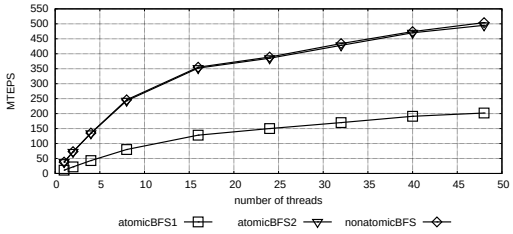
(c) road-europe



(d) friendster

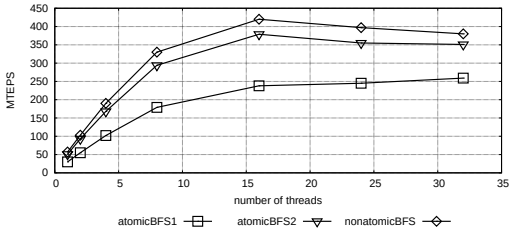


(e) R-100M-2G-45

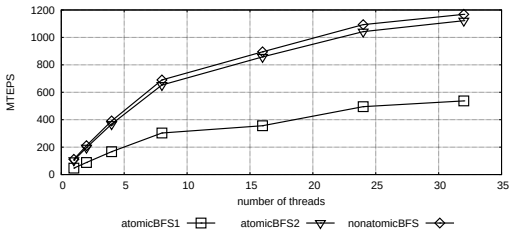


(f) R-100M-2G-57

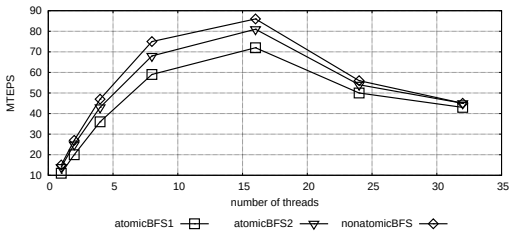
Fig. 10: Performance data on system Intel-IB.



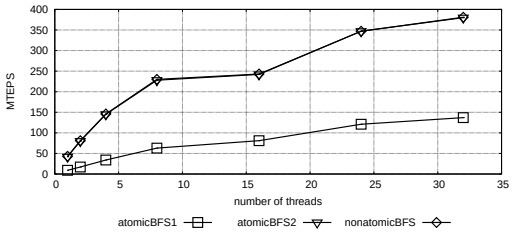
(a) delaunay



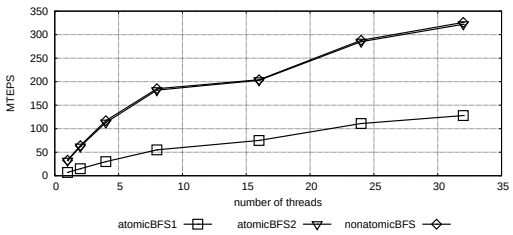
(b) nlpkkt240



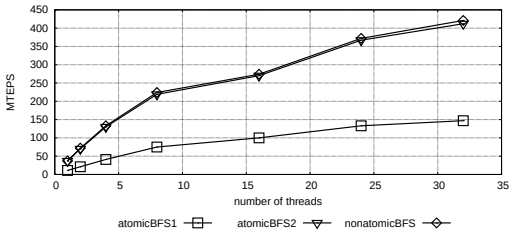
(c) road-europe



(d) friendster

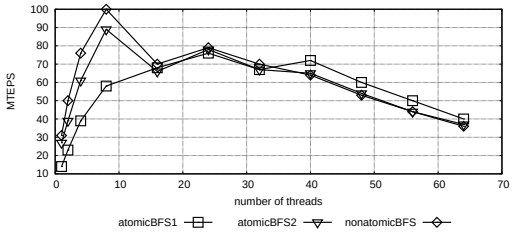


(e) R-100M-2G-45

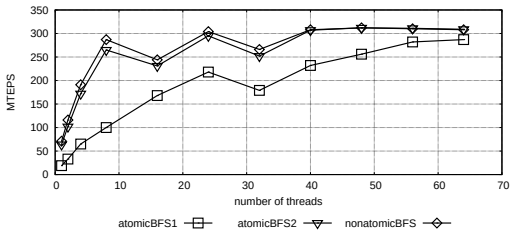


(f) R-100M-2G-57

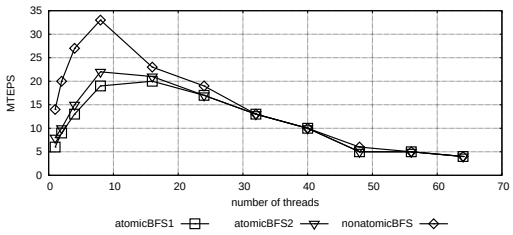
Fig. 11: Performance data on system Intel-SB.



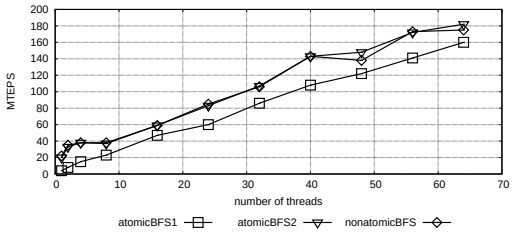
(a) delaunay



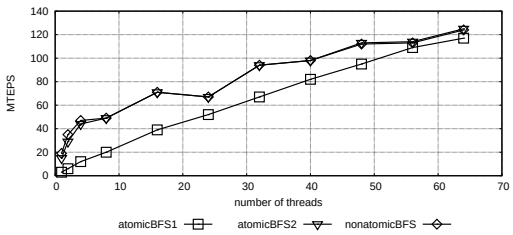
(b) nlpkkt240



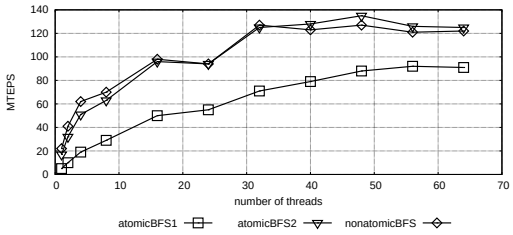
(c) road-europe



(d) friendster

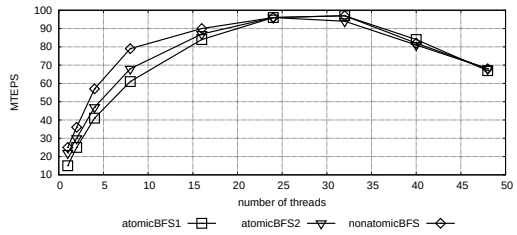


(e) R-100M-2G-45

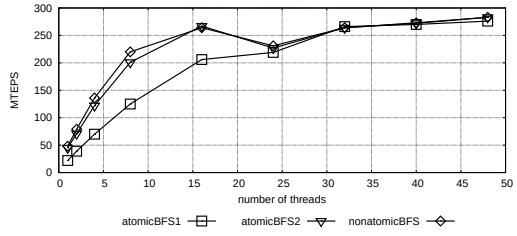


(f) R-100M-2G-57

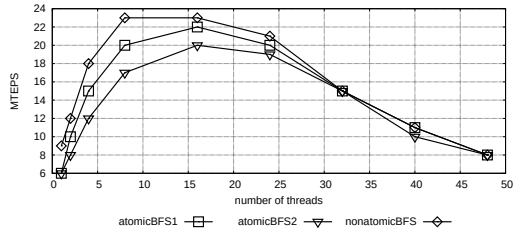
Fig. 12: Performance data on system AMD-IL.



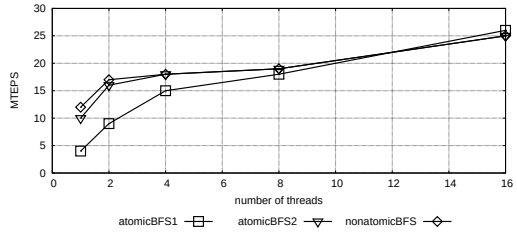
(a) delaunay



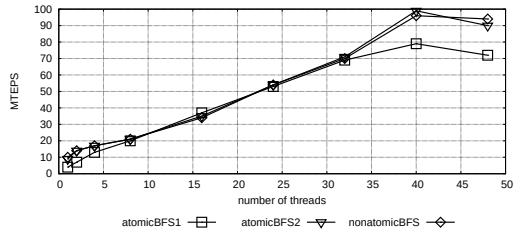
(b) nlpkkt240



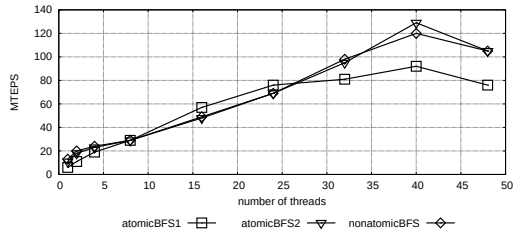
(c) road-europe



(d) friendster



(e) R-100M-2G-45



(f) R-100M-2G-57

Fig. 13: Performance data on system AMD-MC.

accesses. But the smaller the number of threads used, the higher is the difference between the two version in the favour of *nonatomicBFS*. This is because the atomic accesses are the main performance problem, and not memory bandwidth or limited parallelism. Often, the single thread case has the largest relative improvement for the non-atomic version. This is mostly related to the additional overhead for atomic operations (see Section IV) compared to a cheaper non-atomic read and write operation.

On the two AMD systems with 4 CPU sockets the non-atomic version shows more improvements than on the 2-socket Intel systems. The reason for that is the higher overhead for CAS operations on systems with more sockets (see Section IV and especially Fig. 2).

C. Statistical Analysis

In Section VII, the test setup was described. Tests were done on 4 different systems, with different thread numbers depending on the available parallelism of a system, and different input graphs. In total, 761 configurations were tested. So far, selected results were presented in detail. In this section, statistics on all results summarize the performance results.

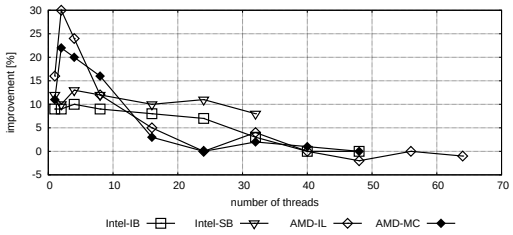
Although all tests were repeated several times (see Section VII for details), there are variations in runtime caused by several and partly non-deterministic factors in a complex parallel system. To leave these artefacts out of the discussion, we define a difference between two results that is below 3% as within the accuracy of measurement and therefore not related to the discussion here.

Fig. 15 shows a statistical analysis of all 761 results. Five classes of test results are shown, given for each system:

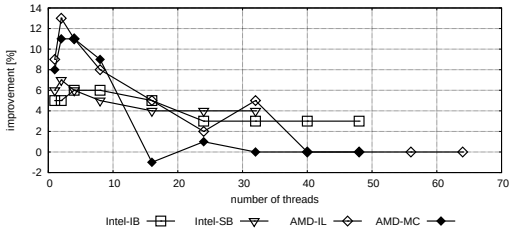
- < -10: where the atomic version *atomicBFS*2 performoms significantly better (better than 10%) than *nonatomicBFS* (in total 2 out of 761 instances)
- $-10 <= x < -3$: where the atomic version *atomicBFS*2 performoms better than *nonatomicBFS* (in total 20 instances)
- $-3 <= x < 3$: where differences are within the accuracy of measurement (in total 406 instances, most of them with large threads counts)
- $3 <= x < 10$: where the non-atomic version *nonatomicBFS* performoms better than *atomicBFS*2 (in total 220 instances)
- $>= 10$: where the non-atomic version *nonatomicBFS* performoms significantly better than *atomicBFS*2 (in total 113 instances)

It can be seen that large improvements (more than 10% performance increase) are mainly on the 4-socket systems while the 2-socket systems show increases that mostly lie below 10%.

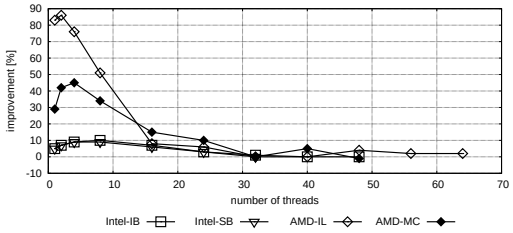
Summarizing the results over all systems, in approx. 2.9% of all test cases the new approach performs worse than *atomicBFS*2, in approx. 43.8% of all test cases the new approach performs better, and in the rest of the test cases the two versions performed rather similar (less than 3% difference). As can be easily seen from Figs. 10 - 13, this is



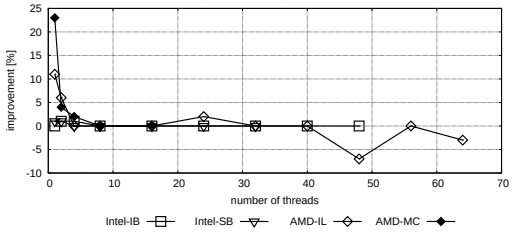
(a) delaunay



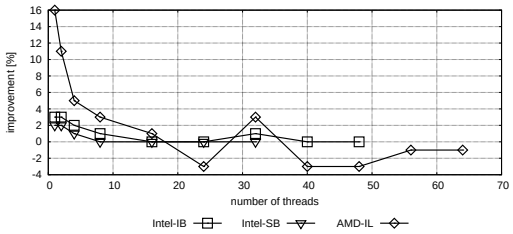
(b) nlpkkt240



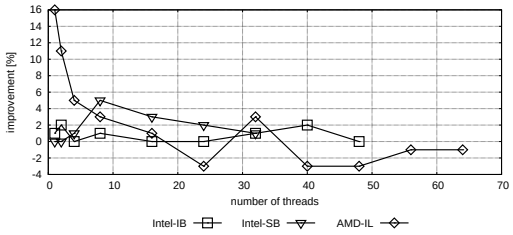
(c) road-europe



(d) friendster



(e) R-100M-4G-45



(f) R-100M-4G-57

Fig. 14: Relative improvements of non-atomic version *nonatomicBFS* compared to optimized atomic version *atomicBFS*2.

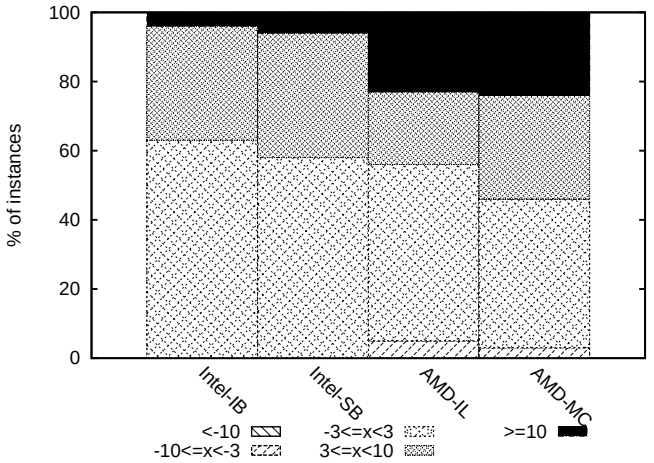
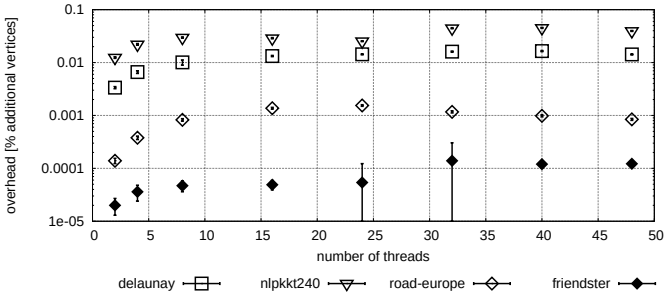
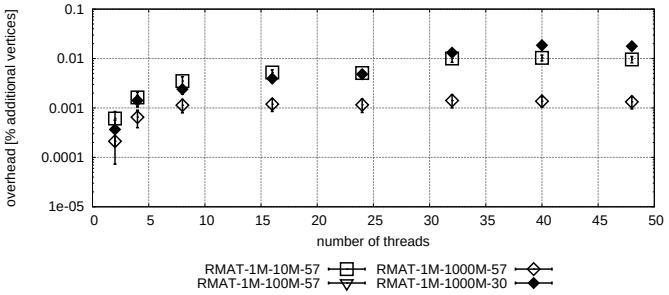


Fig. 15: Classes of performance improvements between *nonatomicBFS* and *atomicBFS*2. $x > 0$ are test instances, where *nonatomicBFS* was faster than *atomicBFS*2).



(a) Percentage of additional vertices (non RMAT graphs)



(b) Percentage of additional vertices (selected RMAT graphs)

Fig. 16: Overhead in percentage of additional vertices for selected graphs on the system Intel-IB, shown with error bars.

often the case when using many threads, where other effects then superimpose the discussion atomic / non-atomic.

D. Additional Overhead

The newly introduced technique avoids atomic operations in exchange with a probability of more work to be executed. To examine the amount of additional work, it was measured how many vertices get inserted multiple times in version *nonatomicBFS*, which is proportional to the additional and redundant work that is generated. The general factors influencing that were discussed already in Section V-B.

Fig. 16 shows exemplarily for the system Intel-IB the overhead for selected graphs including the worst case for that system. The program was run for this test 100 times. Shown in the figure is the average overhead in percent and the standard deviation as an error bar, for a specific number of threads respectively.

As can be seen, the probability increases slightly with more threads, but still this overhead is for the scenario used negligible. Even with 48-fold concurrency, there are very rare situations that lead to multiple insertions. In all tests executed – on all systems, with all number of threads, with all data sets, with different compilers – the additional overhead was in a neglectible range for the application used. In all tests the overhead in percent of *additional* vertices to be handled was always below 0.1 %, and most times even far less than that.

The maximum overhead seen on the system Intel-IB was 0.047 percent, or in absolute numbers, instead of 27,993,600 vertices to be inserted, with $nonatomicBFS$ 29,314,002 vertices were inserted. The difference in this worst case that happened was therefore 13,204 additional vertices. For RMAT-graphs, the difference was even always below 500 additional vertices in absolute values.

IX. Conclusions

In this paper, it was proposed (in parallel programs and within certain scenarios) to replace costly atomic update operations on shared data structures with simple read-write updates. If the correctness of the algorithm is not affected by this change, this leads to an algorithm variant that does not need atomic operations to update the shared data structure. This program variant still works correctly, but on the other side, it may generate more and redundant work to be done.

As an example for such a scenario, a parallel BFS algorithm was used where the atomic detection and update of univisited neighbour vertices was replaced with simple non-atomic read-/write updates. The results for this application show, that the non-atomic version has a huge performance improvement in many situations compared to a straightforward implementation with atomic accesses ($atomicBFS1$). And even compared to an already optimized version using atomic operations only if necessary, the proposed new technique has comparable or many times better performance (approx. 43,9% of all test instances). Especially larger parallel systems with more CPU sockets benefit.

The reason for the performance boost was the avoidance of atomic operations. The additional overhead, which the technique may introduce, was in the BFS application neglectible.

To apply the technique in general, it was shown what requirements must be fulfilled: the test and update operations must be idempotent.

The mainstream transactional memory hardware implementations introduced in recent processors generations (e.g., Intel Haswell) use a different approach. But similar to the approach introduced in this paper, this is an optimistic approach, too, as only the conflict case has to be handled, and not every access. It would be rather interesting to compare these two alternatives with relevant scenarios.

Acknowledgements

The system infrastructure was partially funded by an infrastructure grant of the Ministry for Innovation, Science, Research, and Technology of the state North-Rhine-Westphalia. Matthias Makulla did most of the implementation work on several parallel graph algorithms including an initial version of the ones used in this paper.

References

[1] R. Berrendorf, "Trading redundant work against atomic operations on large shared memory parallel systems," in *Proc. Seventh Intl. Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP)*, 2013, pp. 61–66.

[2] *OpenMP Application Program Interface*, 4th ed., OpenMP Architecture Review Board, http://www.openmp.org/, Jul. 2013, retrieved: 08.03.2014.

[3] IEEE, *Posix.1c (IEEE Std 1003.1c-2008)*, Institute of Electrical and Electronics Engineers, Inc., 2008.

[4] *ISO/IEC 14882:2011 Programming Languages – C++*, ISO, Genf, Schweiz, 2011.

[5] *ISO/IEC 9899:2011 - Programming Languages – C*, ISO, Genf, Schweiz, 2011.

[6] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming*. Burlington, MA: Morgan Kaufmann, 2008.

[7] M. Ben-Ari, *Principles of Concurrent and Distributed Programming*, second edition ed. Harlow, England: Pearson Eduction, 2006.

[8] *Intel©64 and IA-32 Architectures Software Developer's Manual*. Intel Press, 2013, vol. 2: Instruction Set Reference.

[9] *AMD64 Architecture Programmers Manual*. Advanced Micro Devices, 2013, vol. 3: General-Purpose and System Instructions.

[10] *ARM v8 Instruction Set Architecture*. ARM Limited, 2013.

[11] V. Agarwal, F. Petrini, D. Pasetto, and D. A. Bader, "Scalable graph exploration on multicore processors," in *ACM/IEEE Intl.Conf. for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[12] P. E. McKenney, *Synchronization and Scalability in the Macho Multicore Era*, http://www2.rdrop.com/∼paulmck/scalability/paper/ MachoMulticore.2010.08.09a.pdf, retrieved: 27.01.2014.

[13] G. M. Baudet, "Asynchronous iterative methods for multiprocessors," *Journal of the ACM*, vol. 25, no. 2, pp. 226–244, Apr. 1978.

[14] M. Wu, "Asynchronous algorithms for shared memory machines," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1992.

[15] P. Diniz and M. Rinard, "Synchronization transformations for parallel computing," in *Proc. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 1997, pp. 187–200.

[16] D. Novillo, R. Unrau, and J. Schaeffer, "Optimizing mutual exclusion synchronization in explicitly parallel programs," in *Proc. 5th International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, 2000, pp. 128–142.

[17] M. Desnoyers, P. E. McKenney, A. S. Stern, M. R. Dagenais, and J. Walpole, "User-level implementations of read-copy update," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 2, pp. 375– 382, Feb. 2012.

[18] D. Dice, "Implementing fast Java monitors with relaxed locks," in *Proc. JavaTM Virtual Machine and Technology Symposium*, Monterey, 2001, pp. 79–90.

[19] S. Haldar and K. Vidyasankar, "Constructing 1-writer multireader multivalued atomic variables from regular variables," *Journal of the ACM*, vol. 42, no. 1, pp. 186–203, 1995.

[20] M. Herlihy and J. B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *Proc. 20th Intl. Symposium on Computer Architecture*, 1993, pp. 289–300.

[21] A. Wang, M. Gaudet, P. Wu, J. N. Amaral, M. Ohmacht, C. Barton, R. Silbera, and M. Michael, "Evaluation of Blue Gene/Q hardware support for transactional memories," in *Proceedings of the 21st International Conference on Parallel architectures and compilation techniques (PACT'12)*. New York, NY: ACM, 2012, pp. 127–136.

[22] C. Jacobi, T. Siegel, and D. Greiner, "Transactional memory architecture and implementation for IBM system z," in *Proceedings of the IEEE/ACM 45th Annual Intl. Symposium on Microarchitecture*, 2012, pp. 25–36.

[23] J. Reinders, *Transactional Synchronization in Haswell*, http://software.intel.com/en-us/blogs/2012/02/07/ transactional-synchronization-in-haswell, 2012, retrieved 30.01.2014.

[24] L. Lamport, "Concurrent reading and writing," *Journal of the ACM*, vol. 20, no. 11, pp. 906–811, 1977.

[25] J. Aspnes and M. Herlihy, "Wait-free dats structures in the asynchronous PRAM model," in *Proc. 2nd Annual Symposium on Parallel Algorithms and Architectures (SPAA-90)*, Crete, Greece, Jul. 1990, pp. 240–349.

[26] M. Herlihy, "Wait-free synchronization," *ACM Trans. Programming Languages and Systems*, vol. 13, no. 1, pp. 124–149, 1991.

[27] A. LaMarca, "A performance evaluation of lock-free synchronization protocols," in *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, Los Angeles, CA, Aug. 1994, pp. 130–140.

[28] V. Lanin and D. Sasha, "Concurrent set manipulation without locking," in *Proceedings of the 7thACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Austin, TX, Mar. 1988, pp. 211–220.

[29] P. Tsigas and Y. Zhang, "A simple, fast and scalable non-blocking concurrent FIFO queue for shared memory multiprocessor systems," in *Proceedings of the 13th Annual Symposium on Parallel Algorithms and Architectures (SPAA-01)*, Crete, Greece, Sep. 2001, pp. 134–143.

[30] G. Barnes, "Wait-free algorithms for heaps," University of Washington, Seattle, WA, Tech. Rep. TR-94-12-07, 1994.

[31] J. Valois, "Lock-free data structures," Ph.D. dissertation, Rensselaer Polytechnic Institute, Troy, NY, May 1995.

[32] ——, "Lock-free linked lists using compare-and-swap," in *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*, Ottawa, Canada, 1995, pp. 214–222.

[33] H. Sundell and P. Tsigas, "Scalable and lock-free concurrent dictionaries," Chalmers University, Gteborg, Sweden, Tech. Rep. 2003-10, 2003.

[34] H. Sundell, "Efficient and practical non-blocking data stccutures," Ph.D. dissertation, Chalmers University, Gtebord, Sweden, 2004.

[35] K. Fraser and T. Harris, "Concurrent programming without locks," *IEEE Trans. Computers*, vol. 25, no. 2, 2007.

[36] G. Cong and D. A. Bader, "Designing irregular parallel algorithms with mutual exclusion and lock-free protocols," *Journal of Parallel and Distributed Computing*, no. 66, pp. 854–866, 2006.

[37] C. E. Leiserson and T. B. Schardl, "A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers)," in *Proc. 22nd ACM Symp. on Parallelism in Algorithms and Architectures*, 2010, pp. 303–314.

[38] R. Sedgewick, *Algorithms in C++, Part 5: Graph Algorithms*, 3rd ed. Addison-Wesley Professional, 2001.

[39] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[40] C. Wilson, B. Boe, A. Sala, K. Puttaswamy, and B. Zhao, "User interactions in social networks and their implications," in *Eurosys*, 2009, pp. 205–218.

[41] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. Morgan Kaufmann Publishers, Inc., 2012.

[42] Graph 500 Comitee, *Graph 500 Benchmark Suite*, http://www.graph500.org/, retrieved: 08.03.2014.

[43] D. Bader and K. Madduri, "SNAP, small-world network analysis and partitioning: an open-source parallel graph framework for the exploration of large-scale networks," in *22nd IEEE Intl. Symp. on Parallel and Distributed Processing*, 2008, pp. 1–12.

[44] R. Berrendorf and M. Makulla, "Level-synchronous parallel breadth-first search algorithms for multicore- and multiprocessors systems," in *submitted for publication*, 2014.

[45] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, and U. Catalyurek, "A scalable distributed parallel breadth-first search algorithm on BlueGene/L," in *ACM/IEEE Conf. on Supercomputing*, 2005, pp. 25–44.

[46] Y. Xia and V. Prasanna, "Topologically adaptive parallel breadth-first search on multicore processors," in *21st Intl. Conf. on Parallel and Distributed Computing and Systems*, 2009, pp. 1–8.

[47] J. D. Ullman and M. Yannakakis, "High-probability parallel transitive closure algorithms," *SIAM Journal Computing*, vol. 20, no. 1, pp. 100–125, 1991.

[48] S. Beamer, K. Asanovic, and D. Patterson, "Direction-optimizing breadth-first search," in *Proc. Supercomputing 2012*, 2012, pp. 1–10.

[49] J. Chhungani, N. Satish, C. Kim, J. Sewall, and P. Dubey, "Fast and efficient graph traversal algorithm for CPUs: Maximizing single-node efficiency," in *Proc. 26th Intl.Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 378–389.

[50] Y. Yasui, K. Fujusawa, and K. Goto, "NUMA-optimized parallel breadth-first search on multicore single-node system," in *Proc. IEEE Intl. Conference on Big Data*, 2013, pp. 394–402.

[51] G. Taubenfeld, *Synchronization Algorithms and Concurrent Programming*. Harlow, Essex: Pearson Education Limited, 2006.

[52] C. Hoare, "Monitors: An operating system structuring concept," *Comm. ACM*, vol. 17, no. 10, pp. 549–557, 1974.

[53] A. Silberschatz, J. B. Galvin, and G. Gagne, *Operating System Concepts*, 8th ed. John Wiley & Sons Inc, 2008.

[54] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Comm. ACM*, vol. 21, no. 7, pp. 558 – 565, Jul. 1978.

[55] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: A tutorial," *IEEE Computer*, pp. 66–76, Dec. 1996.

[56] M. Dubois, C. Scheurich, and F. A. Briggs, "Synchronization, coherence, and event ordering in multiprocessors," *IEEE Computer*, pp. 9–21, Feb. 1988.

[57] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, "Memory consistency and event ordering in scalable shared-memory multiprocessors," in *Proc. 17th Intl. Symposium on Computer Architecture*. IEEE, 1990, pp. 15–26.

[58] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java Language Specification*, 3rd ed. Addison Wesley, 2005.

[59] DIMACS, *DIMACS'10 Graph Collection*, http://www.cc.gatech.edu/dimacs10/, retrieved: 08.03.2014.

[60] T. Davis and Y. Hu, *Florida Sparse Matrix Collection*, http://www.cise.ufl.edu/research/sparse/matrices/, retrieved: 08.03.2014.

[61] J. Leskovec, *Stanford Large Network Dataset Collection*, http://snap.stanford.edu/data/index.html, retrieved: 08.03.2014.

[62] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *SIAM International Conference on Data Mining*, 2004, pp. 442 – 446.

[63] C. Groër, B. D. Sullivan, and S. Poole, "A mathematical analysis of the R-MAT random graph generator," *Networks*, vol. 58, no. 3, pp. 159–170, Oct. 2011.

Development Framework for Distributed Agile Software Development

Abdullah Saad Alqahtani, John David Moore, David K Harrison, and Bruce M Wood
School of Engineering and Built Environment, Glasgow Caledonian University, Glasgow, United Kingdom
{abdullah.alqahtani, j.d.moore, d.harrison, b.wood} @gcu.ac.uk

***Abstract*—There is a growing interest in applying Agile development methods alongside global software development in order to reap the benefits of both approaches. With this said however, research has shown that software companies are encountering significant challenges when attempting this, due to the contradiction between Agile values and the global development environment. This paper focuses on the challenges encountered with this kind of development, and discusses several techniques via which these challenges can be addressed. It proposes a framework for distributed Agile development. Data has been collected from 85 participants from all around the world using both a self-completed questionnaire and face-to-face interviews. From this study it was found that communication barriers are the biggest development challenge. In order to ameliorate this, development teams and product owners need to work hard to improve the level of communication between them, by adopting a regimented communication schedule. The co-located development strategy "Scrum of Scrums", where distributed isolated teams integrate together using one of the Agile methods, was found to be the most suitable strategy for distributed Agile development.**

Keywords—distributed Agile; development framework; Scrum; Lean and Kanban methods.

I. INTRODUCTION

Increased globalization has led to greater competition between software development companies around the world. The software development industry is seeing a shift from co-located software development to Global Software Development (GSD), which involves multiple distributed development teams in different locations. GSD facilitates competitive software development prices by using teams from countries that have an abundance of IT developers available at relatively low cost [1]. In addition, research has shown that software companies are interested in applying Agile Software Development (ASD) to develop the software by global teams, in order to have the combined advantages of ASD and GSD [2][3]. The combination of Agile development methods and GSD is known as Distributed Agile Software Development (DASD). Venkatesh defines Distributed Agile Development as "*a model in which projects execute an Agile Methodology with teams that are distributed across multiple geographies*" [4]. This combination has shown signs of providing IT companies with the ability to meet the critical success factors of the software industry, such as quality, time, and cost. Sutherland et al. [5] detail their experience of applying a distributed Scrum approach and report several advantages, such as a high increase in team productivity, an increase in the transparency between team members, better building of trust, and increased project visibility. However, although the potential advantages of GSD are clear, research has shown that software companies are encountering significant challenges by applying this approach. Developers are not always able to apply Agile practices successfully due to challenges introduced through the global development environment, including distance and time zone differences [6].

This paper presents the results of a mixed methodology study. The initial, quantitative part aims to study the impact that the projects' settings make to the DASD. Inferential statistics will be used to investigate the differences between the development challenges, which have been reported by the study participants, with regards to the demographic information of their projects' settings. The second, qualitative part involves one specific company, which employs the DASD approach. The study focuses on the challenges of adopting DASD and discusses some possible techniques to address and minimize those challenges. Finally, the results will be integrated to form a development framework to guide users towards a better adoption of DASD.

This paper is structured as follows: first, the related work will be reported. Following this, the research method will be discussed. The results and discussion will be presented in Sections IV and V. Section VI will then address the study validation. Section VI will report the proposed development framework, whilst the final section will contain the summary and conclusion.

II. RELATED WORK

Venkatesh [4] reports some results from surveys by the DH2A Institute. Their data shows that 30% of respondents are using distributed Agile, 40% use local Agile development, and about 85% of them have distributed teams. In addition, the 30% of respondents that stated they are using distributed Agile explained their use of this approach by the advantages and successes they achieved from this kind of development, such as reducing the development cost, accessing the talent pool and resources, increasing team productivity, and decreasing the cost of having high quality software.

However, Software companies are likely to encounter significant challenges and large obstacles when they adopt DASD. Developers may not be able to apply the Agile methods and practices successfully due to the global development environment. The lack of communication and differences in culture and time zones could create huge challenges for Agile methods [7], [8].

A systematic review studied the application of Scrum practices in global software development using 27 literature

studies, and analyzed the challenges into three categories: communication, coordination, and control [9].

The challenges of using Agile with distributed national teams can be categorized into three types of deficiency: communication, trust, and control [3].

Ensuring effective communication within DASD is a huge challenge. The reasons behind these communication challenges could be summarized into four categories: a lack of communication tools, time zone differences, a lack of English language, and a lack of teamwork. These barriers may limit and decrease communication levels in a distributed development [10].

There is a current need for more studies to improve our understanding of how best to adopt Agile methods within global software development. There is a lack of theoretical models of distributed Agile. More studies are needed to address the literature gap by investigating the geographical, cultural, and temporal challenges involved [11].

Previously, we conducted a systematic literature review focusing on the challenges of applying DASD [12]. One of the significant findings of that review was that most of the DASD studies cover the technical perspective of the development but lack coverage of the human perspective. The review also reported that: "The human perspective needs to immediately search to explore the effect of the cultural differences on the relationship between the stakeholders and the development process" [12]. This paper aims to address this issue by exploring the challenges and techniques of applying DASD from the developers' point of view (i.e., the human perspective). Moreover, the systematic literature review classified the development challenges under a four-dimensions model (Communication, Culture, Management, and Agile). This study will provide further investigation into this model and will aim to propose a development framework for this development approach.

III. RESEARCH METHOD

This paper presents a mixed methodology study including both quantitative and qualitative data. This study has been applied to a simple random sample from the study population. According to Kumar [13], the simple random sample is the most popular way to ensure randomization, as each element in the study population has an equal chance of selection. The questionnaire was distributed online to 45 online LinkedIn Agile groups from around the world and many IT organizations were invited to get involved in this investigation. Further collaboration came from two Agile conferences, who sent the questionnaire to their attendees: Lean Agile Scotland Conference 2012 and Agile Worlds - Distributed Agile Conference 2012. Moreover, some software companies agreed to distribute the questionnaire to their staff, although only one global IT development company agreed to let one of its teams be interviewed. All the team members have applied the distributed Agile development and have had a good experience with this kind of development approach.

The questionnaire was configured as a web-based online questionnaire because of the need to gather information from a large number of participants, from a sample study distributed across a number of continents. In addition, the participants were all IT professionals, meaning they were not going to have any difficulties in completing an online questionnaire. The participants were invited to answer the questionnaire online. It ran for five months, from July 2012 to December 2012. Over 120 responses were collected by the end of the data collection stage, including the self-completed questionnaire and the face-to-face interviews; 85 of these were eligible and reliable enough to be investigated. The excluded responses included incomplete responses or responses with "fake" answers (e.g. putting the same answer for all the questionnaire points).

This section addresses the data collection and analysis procedures. It will report the quantitative part first, and then the qualitative one.

A. The quantitative data

The main content of the questionnaire can be classified into two groups of items:

1) *The demographic items – "Categorical variables":* This section aims to collect information about the background of the participants, such as the number of developers involved and the Agile method applied. This section will contribute to a better understanding of the correct setting for the application of the DASD approach.

2) *The development challenges – Likert scales:* This section will use the four-dimensions model, in the form of Likert scales, to evaluate a list of challenges within each dimension. It aims to provide a better understanding of the level of expected issues with this development, by showing what the participants think about the listed points.

This paper will investigate the differences between these responses, with regards to differences between the participants' demographic information. Several null-hypotheses need to be formulated which assume no significant differences between the tested variables until the inferential statistics can prove significant differences. In this case, the alternative hypotheses will be accepted, otherwise the null-hypotheses will be accepted [14]. Estler et al. [15] used the null-hypotheses technique in similar work, in order to study the differences between Agile and structure development. This paper has 10 main null-hypotheses, based on the number of the demographic variables that will be investigated as follow:

H_1: There are no significant differences between the responses of the respondents to the development challenges scales due to the location of the Product owner.

H_2: There are no significant differences in the development challenges scales due to differences in the Number of Developers in the development teams.

H_3: There are no significant differences in the development challenges scales due to the Agile development method adopted.

H_4: There are no significant differences in the development challenges scales due to the Number of different time zone areas amongst development teams.

H_5: There are no significant differences in the development challenges scales due to the Number of distributed teams.

H_6: There are no significant differences in the development challenges scales due to differences in participants' Agile experience.

H_7: There are no significant differences in the development challenges scales due to having development participants from different cultures.

H_8: There are no significant differences in the development challenges scales due to the Number of different time zone areas that the teams are placed into.

H_9: There are no significant differences in the development challenges scales due to differences in the participants' Distributed Agile experience.

H_{10}: There are no significant differences in the development challenges scales due to differences in the applied Distributed development strategy.

Each one of these null-hypotheses has four sub null-hypotheses for each development scale. The development challenges scales data within this study are Likert scale data, so they need to be treated as nonparametric data [16], [17], [18], [19]. The Mann-Whitney U Test needs to be applied to test the differences for the variables with two groups, and the Kruskal Wallis Test for the variables with three groups [14]. Moreover, the median, mean rank and effect size need to be reported to show whether the differences are significant cases or not. Pallant [14] also reports Cohen's criteria [20] to describe the effect size: 0.1 is considered a small effect size, 0.3 a medium effect size and 0.5 a large effect size.

B. The qualitative data

The qualitative data was collected through structured interviews. The interviews were face-to-face and recorded with a voice recorder. Also, notes of the main ideas and answers were taken during the interviews. The data was transcribed from verbal form to textual form. The transferred documents were then compared to the notes from the interviews, to ensure the reliability of the data. Following this, a thematic analysis was applied, which is an approach to identify the themes and patterns from the collected qualitative data [21], [22]. In addition, the data-driven method was selected for the thematic analysis of this study. The data-driven method, regarding to Asnawi, can be summarized in five steps, as follows: "(i) reducing the raw information, (ii) identifying themes within subsamples, (iii) comparing themes across subsamples, (iv) creating a code, and (v) determining the reliability of the code" [23].

The interviews were carried out at a large, global IT development company. The company has 27 offices distributed through 11 countries around the world: Australia, Brazil, Canada, China, Germany, India, Singapore, South Africa, Uganda, the United Kingdom and the United States. The company provides software design and delivery services, as well as development consulting services. It also produces customized software products as tools to support distributed Agile software development, thus helping the development teams to communicate, share information and track progress. The company applies Agile methods in order to develop its global software projects and has been involved in the software industry for the past 20 years. The company requested to remain anonymous within this study.

Three interviewees with good experience of Agile methods and the distributed development approach agreed to participate in this study. Participant-1 had experience working with over 15 teams spanning the entire globe, both East and West, including teams from countries such as India, USA, the UK and Australia. Participant-2 had 4 years of experience, including a special course in Agile development during his master's degree, as well as significant experience when it came to working with stakeholders from different cultures, including people from China, Europe, the UK, the USA and the Middle East. Participant-3 acquired a vast amount of experience before joining the company, since he developed a project while both the product owner and business analyst were away from the development team. He also had experience working with customers from different countries including New Zealand, Australia and the USA.

IV. QUANTITATIVE RESULTS AND DISCUSSION

This section presents the quantitative inferential analysis of the study's null-hypotheses that were reported earlier. The following subsections present the results of each null-hypothesis and its sub-hypotheses in detail, in those cases where there is a significant difference.

A. Product owner location

This demographic variable explores whether the product owner (key stakeholder) was located with one of the development team on-site. It has two answer groups: Yes (i.e. the product owner was located with one of the development teams), and No (i.e. the product owner was not located with one of the development teams).

TABLE I: THE RESULT OF THE MANN-WHITNEY U TEST WITH THE PRODUCT OWNER VARIABLE

	Communication scale	Culture scale	Management scale	Agile scale
Mann-Whitney U	685.000	715.500	772.000	515.000
Wilcoxon W	1813.000	1796.500	1900.000	1596.000
Z	-1.395	-0.781	-0.515	-2.797
Asymp. Sig. (2-tailed)	0.163	0.435	0.606	0.005

Table I above illustrates the U test results. It shows that the difference is highly significant within the Agile scale, with p =0.005 (<=0.01) as a high significant level, so in this case the null-hypothesis for this scale will be rejected and the alternative hypothesis accepted. To investigate the significant differences further, the median and the mean rank are recorded in Table II. The first group has a higher mean rank value of 48.3 while the second group has a value of 34.7. The effect size in this case is r = 0.31. This would be considered a medium effect size, based on Choen's criteria [20].

TABLE II: RANKS FOR THE AGILE SCALE BASED ON THE PRODUCT OWNER

Product owner	N	Mean Rank	Median	Sum of Ranks
No	**34**	48.35	3	1644.00
Yes	**46**	34.70	3	1596.00
Total	**80**			

Table II shows the median and the mean rank. The first group has a higher mean rank value of 48.3 while the second group has a value of 34.7. The effect size in this case is r = 0.31. This would be considered a medium effect size. It is recommended that the product owner is placed onshore with one of the development teams. Having the product owner located on-site with one of the development teams

significantly reduces the risk of any Agile development problems, in terms of the distributed Agile approach. This ensures good levels of communication between the developers and the customers, as well as enabling the development teams to receive rapid feedback. The result shows that having the product owner (key stakeholder) located with one of the on-site development teams could reduce the challenges to the Agile dimension by 31%.

B. Number of developers

This demographic variable has two answer groups: 1) there were 15 developers or fewer within the development team, and 2) there were more than 15 developers within the development team. Table III illustrates the U test results. It shows that the difference is highly significant within the Agile scale, with p =0.006 (<=0.01) as a significant level. In this case, the null-hypothesis for this scale will be rejected and the alternative hypothesis accepted.

TABLE III: THE RESULT OF THE MANN-WHITNEY U TEST WITH THE NUMBER OF DEVELOPERS VARIABLE

	Communication scale	Culture scale	Management scale	Agile scale
Mann-Whitney U	786.000	661.000	671.000	532.500
Wilcoxon W	1689.000	1441.000	1491.000	1312.500
Z	-.542	-1.608	-1.706	-2.766
Asymp. Sig. (2-tailed)	0.588	0.108	0.088	0.006

Table IV reports that the first group has a higher mean rank value of 47 while the second group has a value of 33.6. The effect size in this case is r = 0.30, which would be considered a medium effect size. This indicates that having a greater number of developers does not create more difficulties for the Agile practices. The group with 15 developers or fewer had more Agile dimension issues. 15 members or more had fewer challenges with regard to the Agile dimension – specifically, problems were reduced by 30%. This hypothesis shows that an increase in team members could reduce Agile issues. In other words, a larger number of development participants will help to improve communication and will provide a better environment for Agile adoption. For instance, regarding DASD challenges such as language barriers, members could help each other to communicate better with other team members when they experience communication difficulties. Agile philosophies and values support the idea of having more team members to provide a better environment for sharing information and collaboration as applied with XP programming practices.

TABLE IV: RANKS FOR THE AGILE SCALE BASED ON THE NUMBER OF DEVELOPERS

Number of developers	N	Mean Rank	Median	Sum of Ranks
15 developers or less	41	47.01	3	1927.50
More than 15 developers	39	33.65	3	1312.50
Total	80			

C. Agile methods

There are 7 variables representing the Agile methods, with each variable checking one of the applied development methods (Agile approach in general, Scrum, eXtreme Programming (XP), Scrum/XP hybrid, Lean and Kanban, and Feature-Driven Development (FDD)). Some of the questionnaire participants chose more than one of the Agile methods so this section will investigate each Agile method separately in order to identify any significant differences between the respondents who used or did not use that particular Agile method. The U test results show several significant differences regarding the adopted Agile method. Firstly, the Scrum method has significant differences to the management scale, with p =0.006 (<=0.05) as a significant level. Table V shows the U test results for applying the Scrum approach.

TABLE V: THE RESULT OF THE MANN-WHITNEY U TEST WITH USING SCRUM METHOD

	Communication scale	Culture scale	Management scale	Agile scale
Mann-Whitney U	625.000	655.500	516.500	671.500
Wilcoxon W	1900.000	1880.500	1791.500	1847.500
Z	-1.583	-0.965	-2.734	-0.786
Asymp. Sig. (2-tailed)	0.113	0.335	0.006	0.432

To investigate the significant differences further, the median and the mean rank are recorded in Table VI. The first group has a higher mean rank value of 49.3, while the second group has a value of 35.8. The effect size in this case is r = 0.30 (medium effect size). Applying the Scrum method within the DASD may decrease the risk of challenges arising, with regard to the management and control dimension, by 30%.

TABLE VI: RANKS FOR THE MANAGEMENT SCALE BASED ON USING SCRUM METHOD

Use Scrum method	N	Mean Rank	Median	Sum of Ranks
No	31	49.34	3	1529.50
Yes	50	35.83	2	1791.50
Total	81			

Secondly, the Scrum/XP hybrid method has significant differences to the communication scale, as shown within Table VII (with p =0.006 (<=0.05) as a significant level), and also to the Agile scale (with p =0.03 (<=0.05) as a significant level).

TABLE VII: THE RESULT OF THE MANN-WHITNEY U TEST WITH USING SCRUM/XP HYBRID METHOD

	Communication scale	Culture scale	Management scale	Agile scale
Mann-Whitney U	415.500	558.000	520.500	439.000
Wilcoxon W	2245.500	2269.000	2350.500	2209.000
Z	-2.770	-1.040	-1.588	-2.124
Asymp. Sig. (2-tailed)	0.006	0.298	0.112	0.034

To investigate the significant differences further, the median and the mean rank are recorded in Table VIII and Table IX. The effect size with regards to the communication scale is r = 0.30 (medium effect size), and to the Agile scale it is r = 0.23 (small effect size).

TABLE VIII: RANKS FOR THE COMMUNICATION SCALE BASED ON USING SCRUM/XP HYBRID METHOD

Use Scrum/XP hybrid method	N	Mean Rank	Median	Sum of Ranks
No	60	37.43	3	2245.50
Yes	22	52.61	3	1157.50
Total	82			

The research result suggests that applying the Scrum/XP hybrid method to DASD may increase the challenges faced, with regard to the communication and collaboration dimensions, by 30%. So it is therefore better not to adopt this development method in DASD.

TABLE IX: RANKS FOR THE AGILE SCALE BASED ON USING SCRUM/XP HYBRID METHOD

Use Scrum/XP hybrid method	N	Mean Rank	Median	Sum of Ranks
No	59	37.44	3	2209.00
Yes	21	49.10	3	1031.00
Total	80			

The research result shows that applying the Scrum/XP hybrid method with DASD may increase the challenges with regard to the Agile dimension by 23%. Applying this methodology within DASD is not sufficient where the setting of the distributed development is concerned, so it is therefore better not to adopt this development method with DASD.

Furthermore, Table X illustrates the U test results for the use of the Lean/Kanban method. This development method has significant differences to the culture scale (with p =0.016 (<=0.05) as a significant level), and also to the Agile scale (with p =0.00 (<=0.05) as a significant level).

TABLE X: THE RESULT OF THE MANN-WHITNEY U TEST WITH USING LEAN/KANBAN METHOD

	Communication scale	Culture scale	Management scale	Agile scale
Mann-Whitney U	702.000	529.500	601.500	305.500
Wilcoxon W	1108.000	935.500	1007.500	683.500
Z	-0.572	-2.415	-1.644	-4.490
Asymp. Sig. (2-tailed)	0.568	0.016	0.100	0.000

To investigate these significant differences further, the median and the mean rank are reported in Table XI and Table XII. The effect size with regards to the culture scale is r = 0.26 (small effect size), and to the Agile scale it is r = 0.50 (large effect size).

TABLE XI: RANKS FOR THE CULTURE SCALE BASED ON USING LEAN/KANBAN METHOD

Use Lean/Kanban method	N	Mean Rank	Median	Sum of Ranks
No	52	44.32	3	2304.50
Yes	28	33.41	2	935.50
Total	80			

TABLE XII: RANKS FOR THE AGILE SCALE BASED ON USING LEAN/KANBAN METHOD

Use Lean/Kanban method	N	Mean Rank	Median	Sum of Ranks
No	53	48.24	3	2556.50
Yes	27	25.31	2	683.50
Total	80			

The research result would suggest the use of the Lean/Kanban method to be highly recommended. Applying the Lean/Kanban method within DASD may decrease the challenges to both dimensions: the culture dimension by 26%, and the Agile dimension by 50%.

The U test results show no significant differences within the rest of the development methods, including the Agile approach in general, eXtreme Programming (XP), and Feature-Driven Development (FDD).

The result indicates that there are significant differences from the Agile methods variables for all four scales (communication, culture, management and Agile), so the null-hypotheses will be rejected and all of the alternative hypotheses accepted.

D. Number of time zone areas between development teams

This demographic variable has two answer groups: 1) there are no time zone "spaces" between teams (the "space" [gap] here represents one time zone that has no stakeholders placed in it), and 2) there are one or two time zone spaces between development teams. Table XIII illustrates the U test results. It shows that the difference is significant within the communication scale with p =0.031 (<=0.05) as a significant level, so in this case the null-hypothesis for this scale will be rejected and the alternative hypothesis is accepted.

TABLE XIII: THE RESULT OF THE MANN-WHITNEY U TEST WITH THE NUMBER OF SPACES VARIABLE

	Communication scale	Culture scale	Management scale	Agile scale
Mann-Whitney U	527.500	669.000	692.500	630.000
Wilcoxon W	2123.500	1020.000	2288.500	2170.000
Z	-2.163	-0.409	-0.385	-0.642
Asymp. Sig. (2-tailed)	0.031	0.683	0.700	0.521

To investigate the significant differences further, the median and the mean rank need to be reported. Table XIV shows that the first group has a lower mean rank value of 37.9 while the second group has a value of 49.2. The effect size in this case is r = 0.23. This would be considered a small effect size.

TABLE XIV: RANKS FOR THE COMMUNICATION SCALE BASED ON THE NUMBER OF SPACES

Number of the space between time zones	N	Mean Rank	Median	Sum of Ranks
No space between teams	56	37.92	3	2123.50
There is a space 1 or 2 spaces	26	49.21	3	1279.50
Total	82			

Having no "space" means that there is less than 5 hours time difference between any given development team and

the next closest team, which would mean they have at least 3 overlapping working hours. Having one or two spaces could allow time differences of between 5 and 11 hours. Teams with these time differences could have no overlapping working hours between them, which could bring on many communication challenges to the development. It can increase the challenges with regard to the communication and collaboration dimensions by 23%. It is therefore recommended to maintain a time zone difference of less than 5 hours.

E. Number of distributed teams

This demographic variable has two answer groups presenting the number of distributed teams: 1) less than three distributed teams, and 2) three distributed teams or more.

TABLE XV: THE RESULT OF THE MANN-WHITNEY U TEST WITH THE NUMBER OF DISTRIBUTED TEAMS VARIABLE

	Communication scale	Culture scale	Management scale	Agile scale
Mann-Whitney U	832.500	728.500	749.500	745.500
Wilcoxon W	1693.500	1589.500	1610.500	1606.500
Z	-0.080	-0.824	-0.918	-0.559
Asymp. Sig. (2-tailed)	0.936	0.410	0.358	0.576

Table XV above illustrates the U test results. It shows that there are no significant differences with all the four scales (communication, culture, management and Agile), so the null-hypothesis will be accepted.

F. Agile years of experience

This demographic variable has two answer groups representing the years of experience with Agile: 1) four years of experience or less, and 2) more than four years of experience.

TABLE XVI: THE RESULT OF THE MANN-WHITNEY U TEST WITH THE AGILE YEARS OF EXPERIENCE VARIABLE

	Communication scale	Culture scale	Management scale	Agile scale
Mann-Whitney U	662.500	686.000	674.000	639.500
Wilcoxon W	1158.500	1961.000	2000.000	1914.500
Z	-1.325	-0.767	-1.212	-1.182
Asymp. Sig. (2-tailed)	0.185	0.443	0.225	0.237

Table XVI above illustrates the U test results. It shows that there are no significant differences with all the four scales (communication, culture, management and Agile), so the null-hypothesis will be accepted as below.

G. Multi cultures

This demographic variable has two answer groups: 1) the development teams include participants from different cultures, and 2) the development teams do not include participants from different cultures.

TABLE XVII: THE RESULT OF THE MANN-WHITNEY U TEST WITH THE MULTI-CULTURES VARIABLE

	Communication scale	Culture scale	Management scale	Agile scale
Mann-Whitney U	177.500	148.000	208.500	226.000
Wilcoxon W	205.500	169.000	236.500	254.000
Z	-1.527	-1.631	-0.975	-0.541
Asymp. Sig. (2-tailed)	0.127	0.103	0.330	0.589

Table XVII above illustrates the U test results. It shows that there are no significant differences with all the four scales (communication, culture, management and Agile), so the null-hypothesis will be accepted.

H. Number of time zone areas that the teams are placed into

This demographic variable has two answer groups: 1) all the development teams are placed into one time zone, and 2) the development teams are placed into more than one time zone (2-4).

TABLE XVIII: THE RESULT OF THE MANN-WHITNEY U TEST WITH THE NUMBER OF TIME ZONES VARIABLE

	Communication scale	Culture scale	Management scale	Agile scale
Mann-Whitney U	674.000	660.000	775.500	760.000
Wilcoxon W	1377.000	1695.000	1810.500	1750.000
Z	-1.599	-1.492	-0.0578	-0.333
Asymp. Sig. (2-tailed)	0.110	0.136	0.563	0.739

Table XVIII illustrates the U test results. It shows that there are no significant differences with all the four scales (communication, culture, management and Agile), so the null-hypothesis will be accepted.

I. Distributed Agile experience

This demographic variable has three answer groups, representing the level of distributed Agile experience: 1) less than one year of experience, 2) between one and four years' experience, and 3) more than four years of experience.

TABLE XIX: THE RESULT OF THE KRUSKAL WALLIS TEST WITH THE DISTRIBUTED AGILE EXPERIENCE VARIABLE

	Communication scale	Culture scale	Management scale	Agile scale
Chi-Square	5.222	3.519	3.916	0.927
Df	2	2	2	2
Asymp. Sig.	0.073	0.172	0.141	0.629

Table XIX illustrates the Kruskal Wallis test results. It shows that there are no significant differences with all the four scales (communication, culture, management and Agile), so the null-hypothesis will be accepted.

J. Distributed development strategies

This demographic variable has three answer groups exploring the development strategy: isolated development (teams are distributed geographically but are not cross-

functional), co-located development (distributed isolated teams integrate together using one of the Agile methods), and fully integrated development (teams are distributed geographically and work cross-functionally, using Agile development). Table XX illustrates the Kruskal Wallis test results. It shows that the difference is significant within the management scale, with p =0.038 (<=0.05) as a significant level, so in this case the null-hypothesis for this scale will be rejected and the alternative hypothesis accepted.

TABLE XX: THE RESULT OF THE KRUSKAL WALLIS TEST WITH THE DEVELOPMENT STRATEGY VARIABLE

	Communication scale	Culture scale	Management scale	Agile scale
Chi-Square	0.128	2.623	6.562	0.317
Df	2	2	2	2
Asymp. Sig.	0.938	0.269	0.038	0.854

To investigate this result further, there is a need to apply the Mann-Whitney U test as a follow-up test to identify the differences between each pair groups. The table below shows that the significant differences are between the first and the second group based on the U test results.

TABLE XXI: THE RESULT OF THE MANN-WHITNEY U TEST THE DEVELOPMENT STRATEGY VARIABLE AND THE MANAGEMENT CHALLENGES SCALE

	Management scale
Mann-Whitney U	167.000
Wilcoxon W	728.000
Z	-2.199
Asymp. Sig. (2-tailed)	0.028

Table XXI shows that there are statistically significant differences between the isolated development strategy and the co-located development strategy on the management scale, with p =0.028 (<=0.05) as a significant level which is less than the alpha level of .05. The effect size is considered as a medium effect, with r = 0.31 (medium effect size). Table XXII below shows that the isolated development strategy group has a higher mean rank value of 31, while the second group (co-located development) has 22.

TABLE XXII: RANKS FOR THE MANAGEMENT SCALE BASED ON THE DEVELOPMENT STRATEGY

Development strategy	N	Mean Rank	Median	Sum of Ranks
Isolated	16	31.06	3	497.00
co-located	33	22.06	2	728.00
Total	49			

Having the development teams isolated and not cross-functional may lead to a lack of management and control. Development teams may have problems when combining their developed functions. The study result shows that the isolated development strategy increases the challenges faced, with regard to the management and control dimension, by 31%. The co-located development "Scrum of Scrums" was found to be the most suitable strategy for DASD.

V. QUALITATIVE RESULTS AND DISCUSSION

This section presents the results of the qualitative data by investigating the challenges and the mitigation techniques under the four dimensions model: communications, cultural differences, management and control, and Agile skills

A. Communication and Collaboration Challenges

1) Lack of communication and losing the ability to make immediate decisions (A1): Agile methods require interactive, daily communication among stakeholders. This is difficult to provide within the global environment. The lack of communication and collaboration is a significant issue within the DASD approach [24]. Team members were not able to make immediate decisions, because of the distance between the participants and the lack of communication. As mentioned by Participant-3: "*We lose the ability to have an immediate decision. If we were here at 11am and we wanted to know something straightaway the earliest we could hear from our product owner will be 3pm and that's only if he's got up very early*."

2) Time zone differences (A2): The time zone differences is one of the main reasons that cause DASD's communication challenges [8]. The distance and time zone differences among stakeholders could reduce the available overlap of working hours of distributed teams. Participant-3 reported the issue of having no overlap of working hours by: "*I think if you had two teams where their working days didn't overlap at all, so if you had the UK and the East Coast of Australia where there's something like a 10 hour difference, I don't think that would work*".

3) The lack of English language skills (A3): In most cases, the English language is not the mother tongue of the offshore team members. The lack of proficiency in English could pose a major challenge for the development teams. The different levels of English among the stakeholders could create misunderstandings [25], in the event of people trying to express or indicate meaning by a hint and expecting the others to understand them. Participant-3 reported that: "*If you're having a discussion and there's a thing that you don't say and you assume the other person knows and it's implied, that's where you get the chance for errors*".

Participant-2, who is not a native English Language speaker, described his experience with communication with people with different level of English as hard. Participant-2 stated that: "*The other thing which might be hard is that different people have different levels of English knowledge.*". Also, Participant-2 mentioned some difficulties with understanding native speakers who are speaking with a difficult accent or speaking in a fast way: "*Sometimes it's hard to understand people who are speaking English as their mother language, as well*".

B. Communication and Collaboration Techniques

1) Find a time and a way for synchronised communication (B1): It is important to create an overlap of working hours among the distributed teams. The overlap hours will be used as available time for synchronised communication. Participant-3 reported there should be at least 2 hours of overlapping: "*If you have two teams in different time zones their working days have to have some overlap and if they don't have some overlap and if they don't have some overlap then you need to change the working hours of one of those teams so there is an overlap. I think there needs to be, I would say, at least two hours overlap between those two teams so they can talk face-to-face*".
With some cases that require staying late, the project manager and the business analyst could stay late to communicate with the other stakeholders. Participant-1 stated that: "*PM or BA or whoever needs to showcase something to the client, they need to stay for a while.*".

2) Flexibility regarding working from home (B2): Working hours should be flexible; therefore, the team members should be able to work from home when necessary. This flexibility could help to create overlapping hours among teams. Participant-1 reported that: "*Yeah so the company gives you the opportunity and flexibility to work from home. They also provided the broadband.* ". Participant-2 stated as well: "*The people are free to do and people are getting flexible times to do work from home or work from somewhere else when they are away from the office*".

3) The communication schedule should be regimented (B3): The development stakeholders should have a daily, regimented communication schedule (i.e., Communication Road Map CRM). Such a schedule would help to increase the communication level. Participant-3 reported that: "*I think you need to do what we're doing here at this company and have a very regimented communication schedule*". The product owner should make himself available to communicate with the development team as Participant-3 said: "*I'd say from our product owner's point of view he's got to make sure that he's very involved and he keeps himself aware with what we're up to*". Communication, as reported earlier, is the main issue with the DASD development, so it is necessary to increase the level of communication among the distributed teams. Participant-3 summarised that by: "*You've got to make sure that you communicate well with the stakeholders*".

4) Ask people to speak clearly and be explicit (B4): Regarding the different levels of English skills among the stakeholders, there is a need to speak clearly and to be explicit about what is wanted. Participant-3 mentioned that: "*It's much better to be explicit and to really make clear what you want*".

5) Apply multi-channels for communication (B5): There is a need to have multi-channels for communication. There should be a choice of method and use of the one best suited, such as phone calls, video Skype calls, voice over IP and texting. Participant-2 reported that: "*We are using voice over IPs and the video services. We use Skype, we use GoToMeeting, we have an internal voice over IP device here*", and reported as well: "*we use our own internal service for chatting*". In addition, software to share the screen and knowledge helps teams to share information and increase the visibility of the development. Participant-2 mentioned that: "*So, I can say, tools are really important in distributed systems*".

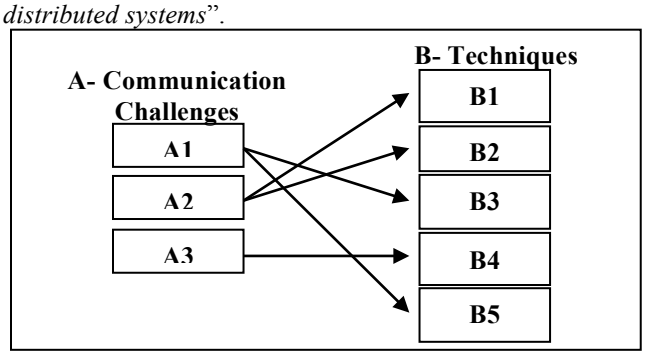


Fig. 1 Communication with DASD challenges and techniques

Fig. 1 illustrates the recommended techniques to address communication and collaboration challenges. It links the challenges with the techniques in order to provide better understanding of them. For example, to address challenge A1, techniques B3 and B5 can be employed. Finally, the CRM must identify a main communication channel that should be selected, based on the project setting and infrastructure. The communication costs with the selected channel need to be considered. The plan should report synchronous and asynchronous channels with an explanation of how to use them.

C. Cultural Differences Challenge

The cultural differences of the stakeholders could create certain misunderstandings [25]. Participant-2 reported that: "*There are a lot of different things in a culture. Like, in some countries, people really like to talk about politics*". Cultural differences could limit the communication between development participants in order to avoid any misuderstandings. Participant-2 stated: "*I feel I know, if somebody from a different culture joins our team, how to behave and then how to find the limits on paid programming, how to speak to people, what sort of questions to ask, what sort of questions not to ask. So, these are the things which we learn*".

D. Techniques to Address the Cultural Differences

1) Creating an open culture within the development teams (D1): There is need to promote an open culture among the project's stakeholders, encouraging people to be free, flexible and liberal. Team members should accept other cultures and try to understand them. Participant-1

mentioned that: "*our culture rules are very liberal, free, there is no dress code. The people are free to do and people are getting flexible times to do work from home or work from somewhere else when are they away from the office. So this flexibility provides a lot of appreciation to the developers and all the people*". Participant-2 also stated that there is need to be flexible within the people from different cultures: "*people who are working in a distributed team, I guess should be more flexible than people who are working on a one - centralised process*".

2) Move the developers between the teams (D2): Providing the team members with the opportunity to move between global offices could help them to discover and explore other cultures. Participant-1 mentioned that by: "*there is a global assignment program which runs every year and it gives a chance to people to work round in any office in the world. So it's a very diverse culture in the company*".

3) A training course for new members (D3): New team members should have a special training course to provide them with the required Agile skills and make them aware of other cultures. The investigated company has a multi-cultural training centre in Bangalore, India. This could help new members to understand different cultures as reported by Participant-1: "*Once you hire anyone, if it's a fresh then he's a graduate. We send them to a university. There is a university which runs in India, in the Bangalore office*". And Participant-1 mentioned as well: "*All the students around the world gather with a different culture in India. They do works together on the same project for three months. After that we send them across different global assignments*".

4) Choose people who fit in with the distributed development culture (D4): Before hiring new people, they should be interviewed to ensure that they fit in with the open culture of the DASD. Participant-1 stated that: "*Always choose the people who actually fit with the culture. We don't choose people who don't fit with the culture*".

In addition, new members should have a qualifying period of a few months, to make sure they fit in with the development culture and environment as reported by Participant-1: "*Even after that, there is a probation of three months, okay. So in the three months itself it is enough time to know the person's attitude and whether - how he is behaving in all the steps. So if he doesn't fit in the culture then we don't extend their assignment*".

5) Flexible working hours and places (D5): This practice was mentioned when addressing communication issues and could also help to increase trust between the company and its employees, one of the cultural issues within the DASD. Participant-1 stated that: "*They do - they know all right that the company the flexibilities providing to them it come with a trust. So the company's putting trust on them so they, of course, need to do the work properly and they also need to put the trust in the company*".

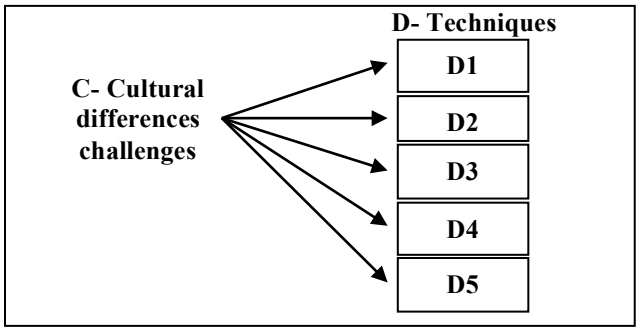


Fig. 2 Cultural differences with DASD challenges and techniques

Fig. 2 illustrates the recommended techniques to address the cultural differences challenges. Techniques D1 to D5 have been applied by the company to minimize the impact of the cultural differences to the development. The cultural differences could reduce the communication as reported early within this section and limit the collaboration between the team members.

E. Management and Control Challenges

1) Updating the developed story on the online wall (E1): Development participants with the DASD approach usually apply an online story wall to track progress. In some cases, they have issues with not updating the developed story on the online wall. This could lead to duplication when developing the required functions/stories. Participant-2 declared that: "*So, sometimes, you - when you get into a story and then it finishes the phase and you start another story, you may forget to move it on the electronic wall*".

2) Estimation difficulties (E2): The second management challenge is with estimation. Large teams could have difficulties with estimating their stories. Participant-2 explained this issue by: "*Estimation for example is one thing that it's hard. So when you have 20 people online and you have 20 people here and you want to estimate stories!!*".

F. Management and Control Techniques

1) Increase communication (F1): There is a need to increase the level of communication in order to manage the work and to resolve any misunderstandings. Participant-3 mentioned that the communication is required to better apply DASD: "*If we do a lot of communication then we can apply all the practice of Agile globally*". Participant-2 reported that as well: "*There should be a lot of communications between the teams as well*". In addition, Participant-1 stated the same thing to manage the distributed Agile development: "*Any company you go there would be the challenge to manage such a vast distributed work, right? It requires a lot of communications; it requires a lot of co-ordinations between all the offices to work together right*".

2) Use software management tools (F2): Using software management tools is required to apply different Agile practices within the distributed development environment. Those tools support the development, make it more visible and easier to track. The tools usually have an online wall for the development stories, which is required to keep it coordinated with the normal story wall. Participant-3 declared that: "*So we have story wall, but that's all replicated in an online tool and we make sure that we keep those two in sync so that the product owner at any time can look at our entire story wall and see what's in progress*". And Participant-2 mentioned the same as well: "*is really important and there are not - and you should be able to first of all, be responsible for updating the electronic wall*".

3) Split large teams (F3): Having a large number of participants could make it difficult to apply some Agile practices, such as estimation. Therefore, splitting large teams could be a solution. However, this requires a lot of communication and coordination between the divided teams. Participant-2 agreed with that by: "*You split the team, you have two PM, you split the number of developers, you will add a new BA. But there should be a lot of communications between the teams as well*".

4) Estimation cards (F4): This practice aims to address the estimation issue. The participants would have a card to estimate each story and they would then show their cards and discuss issues. Participant-2 mentioned that technique by: "*we talk about the story and we count to three and everybody should show a card, or show their hands*".

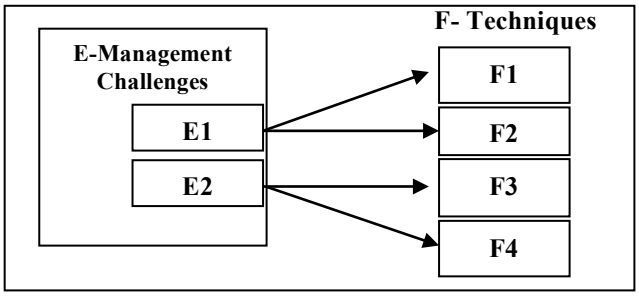


Fig. 3 Managment with DASD challenges and techniques

Fig. 3 links the management challenges with the recommended techniques in order to provide better understanding of them.

G. Agile Level Challenges

1) Lack of a close relationship (G1): The distributed development could result in losing the main aspect of Agile, which is the close relationship between the development participants. Participant-3 mentioned that by: "*I think the main problem with global is - with Agile it's very important to maintain a close relationship to your customers*".

2) Working with traditional organisations/ customers (G2): Traditional organisations/customers may not accept the Agile way of development. They may be used to traditional development approaches, such as the waterfall model [5]. This could decrease the Agility level of the development. For instance, traditional organisations may take their time to allow the developers access to their database or to the necessary information. Participant-2 reported that: "*We speak a lot with tech team, with manager's team, with anyone who can - but they are traditional companies. They have a lot of paperwork for just getting one server, access to one server, or access to a database. But, in an agile company you just ask for something. In our company if you need to access anything…we just ask and we get it as soon as we can. But it's sometimes in other, in client side, in the companies which we are working for they have their own database team which we - a manager should give you permission*". And Participant-2 stated that as well: "*There have been problems with those things. Like database is the obvious one that we can say, you don't get the access to them. You need to go through their process*".

3) Difficulty in applying some Agile practices (G3): The global development setting could make it difficult to apply some Agile practices [22]. For example, the stand up daily meeting is difficult within the distributed Agile development, because of the large number of participants and the lack of visibility among the meeting attendees. Participant-2 reported that by: "*I guess the whole point of stand up is visibility so that you can see somebody and you can ask a question*", and by: "*So imagine if 100 people want to talk for one minute each, it would be a bout two hours while people are standing*".

Furthermore, applying the retrospective practice with the distributed development is difficult as well. Participant-2 stated that: "*Retrospectives are getting affected. Because retrospectives in an agile team are, I guess I feel it's the most physical thing happens because what we do is that we practice different type of RETROS. So what we do is that every iteration that we have RETROS we change them. So we try a lot - because we don't want to make it boring*"

H. Techniques for the Agility Level

1) Use software tools to enable some Agile practices (H1): Usually, development teams adopt various software tools to help them to apply Agile practices. Participant-3 reported that: "*We've done some remote pair programming with him. We use tmux which is a UNIX tool for sharing terminals and we used a VNC client called Chicken and we also use Skype and SSH to set up the connection. So with a combination of those we can have a live pair programming session and that worked quite well*". In addition, Participant-2 stated that as well: "*Tools are really important,, learning how to work with tools are taking time. You may need more efforts*".

2) Dealing with the issues of traditional organisations (H2): Sometimes, IT development companies avoid working with a traditional product owner who is not able to

understand Agile values. Sometimes, they try to provide the traditional product owner with some training about the Agile approach before the project begins. Participant-2 mentioned that: "*So the way that we work is that we try not to accept projects in our company that clients don't give us the chance of working in a way that we want. But some projects it happens that we try - so in some projects when the clients accept that we work for them, but they are not working in agile way. So usually we try to teach, teach the team which we are going to work with them. We communicate a lot, we talk a lot, we have lots of meetings in our team. So we try to settle these things before accepting a project*".

3) Practice for the stand up daily meeting (H3): Practice includes throwing a ball during the meeting. The member who has the ball is the one who is allowed to speak. This practice aims to manage the meeting by allowing one person to speak at a time. In addition, they hold computer tablets, such as iPads, during the meeting to see the distributed members. This practice reported by Participant-2 as: "*we use iPad and we ask them to be online and they talk about it. So we have a ball as a token. We throw it to each other when someone is going to talk.*".

4) Apply simple documentation (H4): One of the techniques in the DASD approach is doing simple reports to share information from the meetings with participants who were not able to attend. Participant-2 declared that by: "*Usually one person writes a simplify - a very simple report that this happens, this decision has been made. This is the reason that we make this decision. So we just read that email every night for example and we get updated about what's happening. If we don't like it, we can state it the day after, or we can send an email and discuss it*".

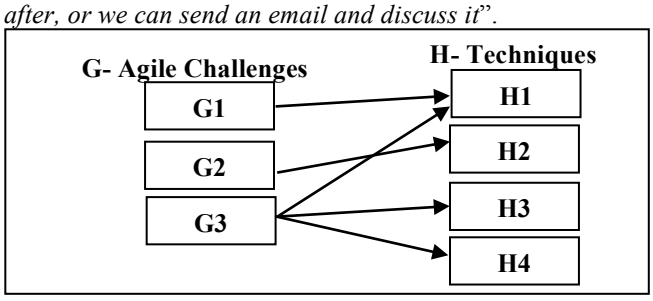


Fig. 4 Agile challenges and techniques with DASD

Fig. 4 reports the Agile challenges and links them with the recommended techniques to award better apply for Agile methods with the distributed development.

VI. VALIDATION

This section addresses the study validation, which could be classified into two categories: the validation of the qualitative data, and the validation of the quantitative data.

A. Validation of the qualitative data

To ensure the validity and the reliability of the study's qualitative analysis and to identify any elements of bias from the researcher, two procedures were applied. Firstly, after the final code was developed, it was tested by other researchers, who applied it to the raw data to ensure that the code and theme analyses were correct. The second procedure was to have the transcripts rigorously checked by other researchers, comparing them to the verbal records and the notes that had been taken. The aim of this was to identify any transcription errors or mistakes [23].

B. Validation of the quantitative data

To ensure the validity of the quantitative data, the questionnaire has been piloted by experts in the field of Agile software development. The aim of this was to check whether the questions are measuring what they aim to measure in an appropriate way. In addition, with multiple-item scales such as the Likert Scale, variables of internal reliability need to be tested. One robust method of achieving that is to apply Cronbach's alpha coefficient. This shows the correlation among all items in the scale. The ideal Cronbach's alpha level is above 0.7 [14]. Table XXIII shows the Cronbach's alpha values for the study scales.

TABLE XXIII: CRONBACH'S ALPHA FOR THE QUESTIONER SCALES

	Scale	Reliability	N of Items
1	The level of communication and collaboration barriers with Distributed Agile Software Development (DASD)	0.888	10
2	The cultural differences (organizational and region culture) barriers with Distributed Agile Software Development (DASD)	0.875	6
3	The lack of management and control within Distributed Agile Software Development (DASD)	0.700	5
4	The lack of Agile with skills within Distributed Agile Software Development (DASD)	0. 748	5
5	Total for the challenges scales	0.765	26

Table XXIII shows the Cronbach's alpha values for each scale. The first four scales represent the challenges scales. The Cronbach's alpha values for those scales are all above 0.7. In addition, the alpha value for the all challenge scales is 0.765. That means the scales and their items are internally consistent.

VII. DEVELOPMENT FRAMEWORK FOR DASD

This section integrates the development strategy of the company under investigation with the study findings in order to provide a standard development framework for DASD. Fig. 5 and Fig. 6 illustrate the proposed framework and it can be organize into four development phases.

The earliest phase is the startup phase, which aims to ensure readiness for the DASD on the development teams' side, and on the product owner's side as well. The communication between product owner and development

teams will be established during this phase. A training course for new members is recommended, as reported within D3. D4 and H2 are also recommended techniques for this phase.

The next stage is the design phase, which includes the gathering of all participants in one place for a few days, to finalize the requirements and estimate the deadline of the project. One of the main techniques that need to be identified within this stage is the design of the CRM. The mitigation techniques for this stage include B1, B3 and B5

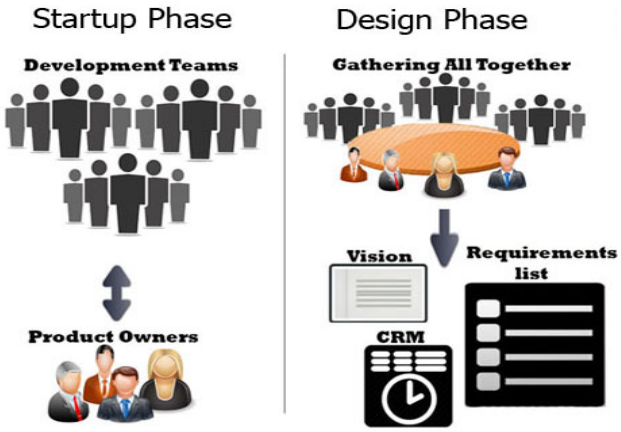


Fig. 5 Startup and Design phases of the development framework

The next phase is the inspection and analysis phase, which aims to break the requirement and development tasks into small development stories. These stories will then be organized and linked to each other. Applying an estimation card can help within this stage, as reported earlier.

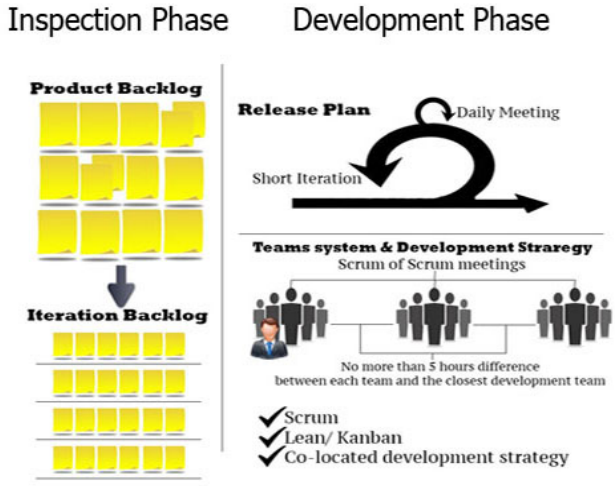


Fig. 6 Inspection and Development phases of the development framework

The final stage is the development phase. Most of the mitigation techniques and study recommendations are linked to this phase, including: D2, F1, F2, F3, F4, H1, H2, and H4. Firstly, teams should have a daily meeting and weekly iteration to increase the level of communication. It is recommended that the product owner be located within one of the development teams. Teams need to have a local meeting and then Scrum of Scrum practice should be applied, to link teams together. In other words, the co-located strategy should be adopted to manage the development teams. In addition, it is recommended to use Scrum or Lean/ Kanban practices as an Agile development method to reduce the development challenges. To reduce the challenges presented by time zone differences, it is not recommended that teams have more than a 5-hour time difference between them and their next closest development team.

Other techniques, such as B2, B4, D1, and D5, are recommended during all the development phases.

VIII. SUMMARY AND CONCLUSION

This paper has presented a mixed methodology study. It has explored the significant differences between the demographic variables and the development challenges scales. The Kruskal Wallis test and the Mann-Whitney U test have been applied as inferential statistics to investigate 10 null-hypotheses. Five of these were rejected and their alternative hypotheses were accepted.

The reported results suggest that communication barriers are the biggest challenge faced when employing the DASD approach. A number of techniques were reported by the participants to address the known communication issues affecting this approach. Most of the issues related to the lack of communication between stakeholders. The development teams and product owners need to work hard to increase the level of the communication between them.

The other main issue encountered was the lack of Agile skills and knowledge from the developers and the product owners. A global setting makes this issue all the more pronounced, because of the distance between the stakeholders. In these cases, there is a need to improve knowledge of Agile by providing training courses and Agile coaching, to ensure the sufficient application of Agile practices.

The management issues are also related to the distance and the size of the development teams. Improving the communication level and Agile skills could reduce these management difficulties. Splitting teams may be with a solution for teams with a large number of developers.

The issue of cultural differences is the least important problem because most of the stakeholders are aware of each other's cultures and have the ability to work with different people. However, some misunderstanding could arise, particularly through a lack of communication. Thus, it is essential that the development participants are clear, flexible, and open with those from other cultures. The experience with DASD from the investigated company helped develop a clearer understanding of the challenges of cultural differences. The application of techniques such as training courses can help to minimize these cultural differences issues. Moving the team members between the various development teams throughout the world will also help them

to better understand other cultures and could address this issue.

In conclusion, this case study has highlighted some of the major challenges of applying DASD. It has also proposed a development framework that outlined several development practices to achieve a more effective application of this development approach. The discussion has shown that the study findings are in agreement with existing literature on most of the investigated points.

Future work will involve further investigation into the proposed framework in order to understand its impact on the DASD approach.

IX. ACKNOWLEDGMENT

This work is supported by the University of Dammam, the Ministry of Higher Education, Kingdom of Saudi Arabia, (PhD scholarship).

REFERENCES

[1] A.S. Alqahtani, J. Moore, D. Harrison, and B. Wood, "Distributed Agile software development challenges and mitigation techniques: A Case Study," ICSEA 2013, The Eighth International Conference on Software Engineering Advances, 2013, pp. 352-358.

[2] M. Nisat and T. Hameed, "Agile methods handling offshore software development issues," 8th International: Multitopic Conference, Proceedings of INMIC, 2004, pp. 417-422.

[3] A. Szőke, "Optimized feature distribution in distributed agile environments," Product-focused software process improvement, 2010, pp. 62-76.

[4] U. Venkatesh, Distributed Agile: DH2A - The proven Agile software development approach and toolkit for geographically dispersed teams. New Jersey: Technics publications LLC., 2011.

[5] J. Sutherland, G. Schoonheim, N. Kumar, V. Pandey, and S. Vishal, "Fully distributed scrum: Linear scalability of production between San Francisco and India," Agile conference, IEEE, 2009, pp. 277-344.

[6] E. Hossain, M. Babar, and H. Paik, "Using Scrum in global software development: A systematic literature review," Fourth IEEE International Conference on Global Software Engineering, 2009, pp.175-184.

[7] S. Modi, and P. Abbott, "Understanding Collaborative Practices in Distributed Agile Development: Research Proposal," 8th International Conference Global Software Engineering Workshops (ICGSEW), IEEE, 2013, pp. 74-77.

[8] S. Jalali, and C. Wohlin, "Global Software Engineering and Agile Practices: A systematic review," Journal of Software Maintenance and Evolution: Research and Practice. 2011, pp. 643–659.

[9] E. Hossain, P. Bannerman, and D. Jeffery, "Scrum practices in global software development: A research framework," Product-focused software process improvement, pp. 88-102., 2011.

[10] S. Dorairaj, J. Noble, and P. Malik, "Bridging cultural differences: A grounded theory perspective," Proceedings of the 4th India Software Engineering Conference, ACM, 2011, pp. 3-10.

[11] D. Smite, N.B. Moe, and P.J. Agerfalk, "Agility Across Time and Space: Summing up and Planning for the Future," Agility Across Time and Space. Springer Berlin Heidelberg, 2010, pp. 333-337.

[12] A.S. Alqahtani, J. Moore, D. Harrison, and B. Wood, "The challenges of applying distributed Agile software development: A systematic review," International Journal of Advances in Engineering & Technology, Vol. 5, Issue 2, 2013, pp. 23-36.

[13] R. Kumar, Research methodology: A step-by-step guide for beginners. 4th ed. SAGE publications, 1 Olive's Yard, 55 City Road, London. 2014.

[14] J. Pallant, The SPSS survival manual: A step by step guide to data analysis using IBM SPSS. Open University Press McGraw-Hill Education, England, 2013.

[15] H. Estler, M. Nordio, C.A. Furia, B. Meyer, and J. Schneider, "Agile vs. structured distributed software development: A case study", Global Software Engineering (ICGSE), IEEE Seventh International Conference, 2012, pp. 11-20.

[16] A. Garth, Analysing data using SPSS: A practical guide for those unfortunate enough to have to actually do it, Sheffield Hallam University. 2008.

[17] S. Jamieson, "Likert scales: how to (ab) use them", Medical education, vol. 38, no. 12, 2004, pp. 1217-1218.

[18] D. Bertram, "Likert scales", Matematicki fakultet, University of Beogradu, 2007.

[19] J.R. Boone, and D.A. Boone, "Analyzing Likert Data", Journal of Extension, vol. 50, no. 2. 2012.

[20] J.W. Cohen, Statistical power analysis for the behavioral sciences. Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.

[21] R. Boyatzis, Transforming qualitative information: Thematic analysis and code development. SAGE publications incorporated. 1998.

[22] C. Dawson, Introduction to research methods: A practical guide for anyone undertaking a research project. Oxford: How To Books Ltd. 2009.

[23] A. Asnawi, A. Gravell, and G. Wills, "Emergence of Agile methods: Perceptions from software practitioners in Malaysia," AGILE India, 2012, pp. 30-39.

[24] M. Paasivaara and C. Lassenius, "Using Scrum Practices in GSD Projects," Agility Across Time and Space. Springer Berlin Heidelberg, 2010. pp. 259-278.

[25] M. Kajko-Mattsson, G. Azizyan, and M.K. Magarian, "Classes of distributed agile development problems," The Agile 2010 Conference, IEEE, 2010, pp. 51-58.

CREATE: A Co-Modeling Approach for Scenario-based Requirements and Component-based Architectures - A Detailed View

Björn Schindler, Marcel Ibe, Martin Vogel, and Andreas Rausch
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
{bjoern.schindler, marcel.ibe, m.vogel, andreas.rausch}@tu-clausthal.de

***Abstract*—Requirements engineering and architectural design are key activities for successful development of software-intensive systems and are strongly interrelated. Particularly, in early development stages requirements and architecture decisions are frequently changing. The fundamental problem addressed in this paper is the development of inconsistencies at the iterative evolution of requirements and architectures. Inconsistencies between requirements and architectures lead to an incorrect consideration of requirements by the system under development and consequently to unfulfilled requirements. Thus, advanced systematic approaches are needed, which could minimize the risks of wrong early decisions during the iterative evolution of requirements and architectures. Our model-based approach supports the iterative evolution of requirements and architectures by defining a concrete description technique. It provides simplified scenario-based models for a precise description of requirements, which are suitable for validation by stakeholders. Furthermore, the approach provides a component-based model for a precise and entire description of architectures. Strict interrelations between scenario-based and component-based models support the consistence maintenance. These interrelations enable even an automation of this task. In this paper, the model-based approach CREATE is described in all details. For example, all interrelations are introduced completely.**

Keywords-requirements; architecture; evolution; consistency.

I. INTRODUCTION

Requirements Engineering (RE) and Architectural Design (AD) are essential for successfully developing high-quality software-intensive systems [1]. RE and AD activities are intertwined and iteratively performed [2]. The architecture of a software system must satisfy its requirements. In practice, architectural constraints frequently prohibit an entire realization of all requirements. This might imply a change to the initial requirements or the selection of a different appropriate architecture. Further, additional requirements might be discovered during the development process, leading to changes in the architecture. Design decisions that are made early in this iterative process are the most crucial ones, because they are very hard and costly to change later in the development process.

In classical development processes (e.g., the waterfall model [3]), artifacts like, for instance, the requirements specification or the architecture are developed sequentially. This is also the case at iterative process models like the spiral life cycle model of Böhm [4]. The iterative, concurrent evolution of requirements and architectures demands that the development of an architecture is based on incomplete requirements. Also, certain requirements can only be understood after modeling or even partially implementing the system architecture. Nuseibeh [2] describes an advanced approach, which adapts the spiral life cycle model and aims at overcoming the often artificial separation of requirements specification and design by intertwining these activities in an iterative evolutionary software development process. This approach is called the twin peaks model. To map requirements onto architectures and maintain the consistency and traceability between the two Grünbacher et al. [5] introduces an intermediate model called Component Bus System Property (CBSP) (see Fig. 1). This model maps requirements to architecture elements by the CBSP model, which allows a systematic way to reconcile requirements with stakeholders.

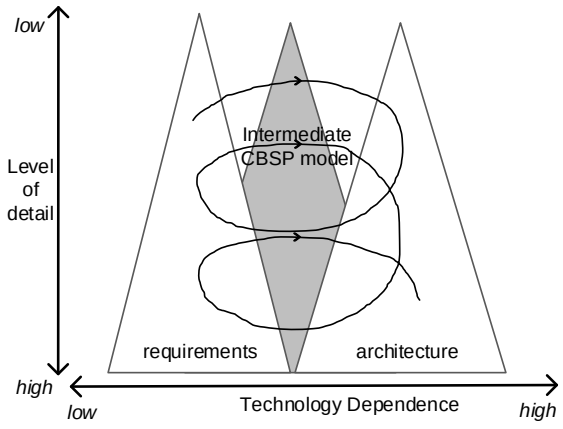


Figure 1. Intermediate model within the twin peaks [5]

Nevertheless, the advanced twin peaks model is kept very general. For instance, it does not specify the level of detail of requirements in relation to the architecture [6]. Due to the fact that there is no concrete advanced approach supporting the iterative evolution of requirements and architecture we were commissioned by the german armed forces and the german government to undertake a research project [1]. In order to be able to consider all required aspects, we made an expert survey. Therefore, we interviewed staff and leaders of three medium to big sized development projects with up to 30 project participants on customers and contractors side about their problems in the field of RE and AD.

A general mentioned problem was that the developed systems did not fulfill all requirements of the customers. The result of the survey was a list of the following reasons and derived guidelines:

- For the contractor the requirements were too informal, imprecise and incomplete. Requirements had to be repeatedly elicited and specified during architecture design. Hence, requirements on a software system should be complete and precise.
- At the elicitation process, the reconcilement of more precise and formal descriptions was too costly. The reason was the need of a detailed explanation by the contractor. For an improved reconcilement requirements descriptions should be precise as well as comprehensible. These guidelines are also mentioned by Nuseibeh [7]. Furthermore, the complexity of the models have to be manageable for validation by stakeholders.
- The most serious problem was caused by frequently changing requirements during architecture design. These changes cause inconsistencies between requirements and architectures. Thus, many requirements were not fulfilled by the developed systems. In consequence, an architecture should not only describe the system by a definition of its structure and behavior. It should also describe precisely how the system under development fulfills the given requirements. Consistency constraints between requirements and architectures should be defined, which support the consistency maintenance.

Starting from this initial situation the target of the project was the development of a domain specific model-based approach, which fulfills the mentioned guidelines. The result of the project was the model-based approach CREATE [1]. The main goal of CREATE is to provide a concrete description technique for requirements and architectures, which supports the iterative evolution in the sense of twin peaks. Requirements descriptions have to be precise and comprehensible. This necessitates a well-balanced trade-off between expressiveness and manageability of models for the description of requirements. This trade-off is achieved by providing simplified scenario-based requirements models. Furthermore, CREATE provides a component-based architecture model for a precise desciption of how the system under development fulfills the given requirements. A definition of strict interrelations support the iterative evolution of requirements and architectures. Complex interrelations between requirements and architectures cause a high complexity for consistence maintenance. Thus, CREATE proposes interrelations between requirements and architecture models, which concretize a well-balanced trade-off between expressiveness and manageability.

In this paper, the CREATE approach is described in all details. Furthermore, the experiences at the application in practice are described. The presented approach is domain-specific, since it is developed for information systems like web-based systems and modern communication systems. In Section II, existing model-based approaches for the iterative evolution of requirements and architectures are considered. In Section III, the overall approach is introduced and in Section IV the description technique is described in detail at an example. The support of the consistency maintenance is described in Section V. Section VI contains a description of our experiences at the development and application of the approach in practice. Section VII includes a discussion of the results and pending points for future work.

II. Related Work

Existing model-based development approaches for requirements and architectures can be categorized into model-based approaches for requirements engineering, model-based approaches for architecture design and combined approaches.

Representative model-based approaches for requirements engineering are described in [8]–[10]. In [8], requirements are described by Unified Modeling Language (UML) [11] activity diagrams. A formal operational semantics enables execution of activity diagram specifications. The executed activity diagram specification serves as prototype for visualization of requirements. In the approach illustrated in [9], UML collaboration diagrams are enriched by user interface information in order to specify elicited requirements. These diagrams are transformed into complete dynamic specifications of user interface objects represented by state diagrams. These state diagrams are used for generation of prototypes. In [10], use case and user interface information are recorded at stakeholder interviews. Therefore, use case steps are enriched by scribbled dialog mockups. Prototypes are created, which visualize dialog mockups of use case steps in sequence for fast feedback of stakeholders. In general, these approaches have a well elaborated model structure for requirements engineering and improve the validation of requirements by stakeholders. On the other side, the mapping to the architecture is not precisely enough defined at these approaches to support a iterative evolution of requirements and architectures.

Representative approaches defining models for architectural design are described in [12] and [13]. Model-Driven Architecture (MDA) [12] is a framework for software development. In MDA, the Computation Independent Model (CIM) can be used to describe business processes. The Platform Independent Model (PIM) may describe the structure and behavior of the software system. Component models like KobrA [13] are concrete model-based approaches based on MDA. In general, these approaches have a well elaborated model structure for architecture design and enable a detailed description of the structure and behavior of the software system. On the other side, these approaches do not support an iterative evolution of requirements and architectures. The mapping between requirements and architectures is not precisely enough defined for this field of application.

Representative combined modeling approaches for requirements and architectures are described in [14], [5], and [15]. In [14], a Requirements Definition Language (RDL) is used, which allows a structured definition of requirements. Meta model elements of the RDL are mapped to corresponding meta model elements of the Architecture Description Language (ADL). The approach described in [5] uses the intermediate model CBSP to map requirements to architecture elements. Different subtypes of CBSP elements allow classification of

requirements. Requirements exhibit overlapping CBSP properties can be split and refined until no stakeholder conflicts exist. The Software Architecture Analysis Method (SAAM) [15] describes a method for a scenario-based analysis of software architectures. SAAM defines also the activities of the scenario-based analysis. In SAAM, scenarios and architecture descriptions are developed iteratively [15]. For each scenario it is determined whether a change of the architecture is required for execution. Based on the importance and conflicts of required changes an overall ranking of the developed scenarios is determined. An advantage of these approaches is the combination of techniques for the description of requirements and architectures. On the other side, these approaches are very abstract and do not specify concrete models and mappings, which fulfill the conditions defined in the introduction for an adequate description of requirements and architectures.

Besides the stated existing approaches further approaches are conceivable, which are based on synthesis approaches [16] of complete state-based models from scenario-based models. Scenario-based and state-based models can potentially be used for the description of requirements and architectures. Consistency is, for instance, a subject of the approaches described in [17] and [18]. Unfortunately, these approaches are generally maintaining a complete consistency by means of a bijection. Architectures need to describe more details about the software system. These details have to be well separated from the requirements. Hence, an alternating correction of inconsistencies and not a bijection is required for the support of a iterative evolution of requirements and architectures.

Other approaches are focusing on the consistence maintenance of models. Representative approaches of this kind are described in [21], [22]. In [22] an approach for the automatic consistence check of behavioral requirements and design models is described. The approach introduced in [21] allows an automatic consistence check of general UML models. These approaches focus on the consistence maintenance and not on the provision of a concrete description technique for requirements and architectures. In consequence, these approaches do not provide a complete set of concrete models and mappings, which fulfill the conditions defined in the introduction for an adequate description of requirements and architectures.

The main goal of CREATE is to provide a concrete description technique for requirements and architectures, which supports the iterative evolution by fulfilling all guidelines mentioned in the introduction. It defines a complete set of concrete models for the description of requirements and architectures. Further, it defines concrete interrelations between these models, which support the consistence maintenance.

III. Overall Approach

Our domain specific model-based approach supports concurrent development of requirements and architectures. An appropriate process for concurrent development is described by the twin peaks model [2]. In this model, requirements and architectures have an equal status and are evolved iteratively. This is illustrated by twin peaks (see Fig. 2).

Our domain specific model-based approach concretizes twin peaks by defining a concrete description technique. Diagrams

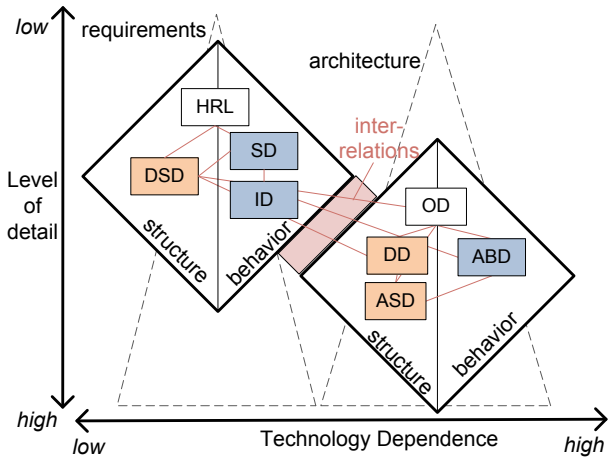


Figure 2. CREATE approach within twin peaks

are used for a precise description of requirements and architectures. These diagrams are illustrated within diamonds in the twin peaks model (see Fig. 2). The process flow of our approach begins with a formal description of inital requirements. Afterwards, the architecture is developed and consistency to the requirements is maintained continuously. Inconsistencies are resolved by changing requirements or the architecture.

The main contribution of CREATE is the concrete description technique with defined interrelations between requirements and architecture descriptions. It is well known that scenarios help to elicit and validate requirements [16]. A precise description of elicited requirements can be achieved by scenario-based models [16]. The co-modeling approach provides simplified scenario-based models for the description of requirements. Furthermore, the description is reduced to representative and concrete scenarios. Hence, the complexity of these models is manageable for the validation by stakeholders. The validation is improved by combining these models with models enabling visualization of requirements by user interface mockups [10]. An architecture specified by CREATE describes the behavior and the resulting structure of the software system precisely. This description is supported by a component-model. Component-Based Software Engineering (CBSE) [19] has been continuously improved and successfully applied over the past years. Systems are composed by existing software 'parts' called software components. Component models enable a precise description of component-based architectures [20].

In our domain specific model-based approach, diagrams are used to model structural or behavioral aspects of requirements and architectures. The domain structure (e.g., business structure) defines important requirements on the system. In CREATE, it can be described by a Domain Structure Diagram (DSD), which is assigned to the structure part of the requirements diamond (see Fig. 2). Elicitation and specification of processes at the domain (e.g., business processes) is an important aspect at requirements engineering. In our approach, these processes can be described by a Scenario Diagram (SD)

in combination with an Interaction Mockup Diagram (ID). The SD and ID are assigned to the behavior part of the requirements diamond. Additional text-based requirements can be captured in the Hierarchical Requirements List (HRL). These texts can describe structural as well as behavioral aspects. CREATE diagrams have not to be developed in a strict order. Typically, rough text-based requirements are captured first. The DSD as well as the SD and ID are, in general, evolved iterativly beginning with a rough first scenario. At the architecture design, the definition of the system boundary and the provided functions is crucial. These aspects of the architecture can be described precisely by the System Overview Diagram (OD) of the CREATE architecture (see Fig. 2). The process of a function is described by the Architectural Behavior Diagram (ABD). Data objects used in these processes are defined in the Data Diagram (DD). A suitable structure of the system can be derived from the process descriptions. This structure is described by the Architectural Structure Diagram (ASD). The diagrams of the CREATE architecture are iteratively developed as well. Typically, a small set of system functions are described by the OD, ABD and DD first. Afterwards, the ASD is derived, and finally, additional system functions are described. During the development of the architecture, consistency to the requirements is maintained continuously. In Section IV, the description technique is described in all details.

Existing languages, such as UML [11], include among others structural and behavioral diagrams for the modeling of systems. In this paper, CREATE uses exemplarily a subset of UML diagrams and their available model elements to formally describe requirements and architectures. Additional diagrams are used to enable a visualization of requirements by user interface mockups. Interrelations between these diagrams are precisely defined. Consistency maintenance during the development of requirements and architectures is supported by defined interrelations between scenario-based requirements models and component-based architecture models (see Fig. 2). Interrelations are also defined within these models. One interrelation between requirements and architectures is, for instance, that the system boundary of the OD represents a part of the domain structure in the DSD. An examplary interrelation within the requirements description is that every object used in a scenario has to be a part in the domain structure. In Section V, all interrelations are described in detail. Interrelations are defined by constraints. The definition of these constraints support the consistency maintenance, because they can easily be checked. Furthermore, they enable an automatization of the consistency maintenance [24].

IV. Description Technique

In this section, details of the description technique are described by a case study. The subject of this study is the development of a library system. Requirements on the library system are described by the scenario-based model of CREATE, which is suitable for the validation by stakeholders. The architecture of the library system is described by the CREATE component model. This component model allows the description of the behavior of the system under development and the resulting internal structure. In the following, initial requirements on the library system are described by the provided diagrams. Afterwards, the architecture of the system is described. Based on the requirements and the architecture of the library system the description technique of CREATE is explained in detail.

A. Requirements Description

In CREATE, requirements on a system are precisely described by a scenario-based model. The scenario-based model of CREATE consists of the provided SD, ID and DSD, which are the core diagrams of the requirements description. The domain structure like the business structure defines important requirements on the system. In CREATE, it can be described by the DSD. Processes on this domain like business processes are described by the SD in combination with ID. Since requirements are frequently mentioned text-based (e.g., in protocols or meetings) the HRL is provided. The HRL allows the capturing of text-based requirements. A strict order for the development of the CREATE diagrams is not prescribed. In practice, rough text-based requirements are elicited initially. Afterwards, the scenario-based model is developed. A rough first scenario is refined by an iterative development of the DSD, the SD and the ID. In the following, the HRL is explained by exemplary text-based requirements on the library system. Afterwards, the scenario-based model is explained in all details. For this detailed explanation the DSD, the SD and the ID are used for the definition of precise requirements on the library system.

1) HRL: Requirements are frequently mentioned text-based (e.g., in protocols or meetings). The HRL allows the capturing of text-based requirements. In a requriements specification, several HRLs can be introduced in ordner to distinguish between different classes of requirements. In our case study, we introduce two HRLs for the capturing of functional and non-functional requirements (see Fig. 3). Contents of the HRL can be structured hierarchical. In this way, it is possible to refine one requirement by several other requirements. In our example, the HRL *functional requirements* contains, for instance, the requirement *2*. This requirement demands that the system must be able to process requests for book orders of users. This is refined by requirement *2.1*, which demands that a user should be able to send a book order request to the manager by the library system.

2) DSD: In general, the system under development has to be integrated in a domain structure (e.g., a business structure). The domain structure defines important requirements on the system. The DSD allows a precise description of the domain structure and is based on the UML Composition Structure Diagram [11]. The most important modelling elements of the DSD are parts and connectors [11]. Parts are used to describe the system under development, external systems, persons and entities of the domain. Every part must have a type. In return, one type can be used for several parts. In our example, the parts *LibrarySystem* and *Printer* represent systems (see Fig. 4). The multiplicity defines the minimal and maximal number of objects in this part. According to the DSD, the library domain contains, for instance, exactly one library system. The box that represents the part *LibrarySystem* is filled gray to mark it as

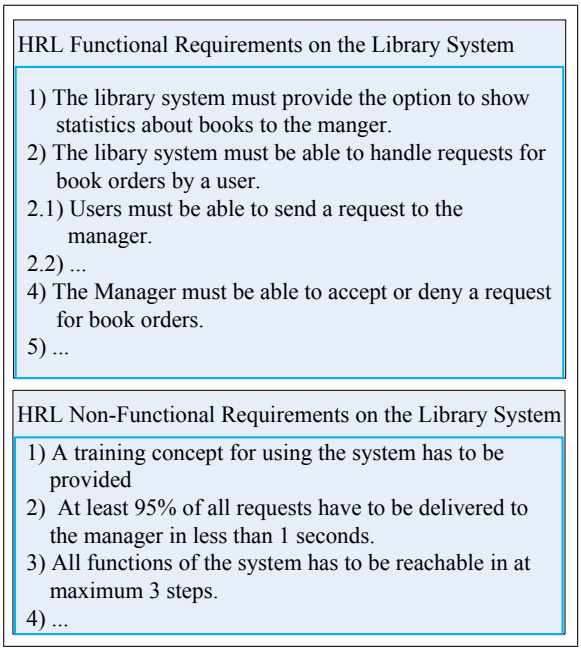


Figure 3. Hierarchical Requirements List of the Library System

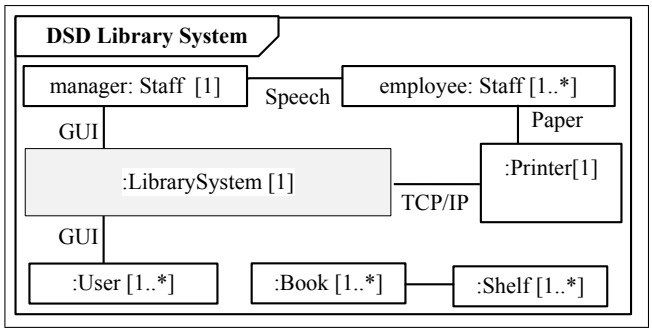


Figure 4. Domain Structure Diagram of the Library System

the system under development. Three parts in the domain are representing persons: *User*, *Manager* and *Employee*. *Manager* and *Employee* are both of the type *Staff*. The parts *Book* and *Shelf* describe typical domain entities. Connectors describe the ability of two connected parts to interact with each other. A connection between two parts means that every object of one part can interact with all objects of the other part. It is also possible, to define the type of the connector. This type describes the kind of the interaction in more detail. The domain structure of the library system describes several connectors between parts, which specify the kind of interaction. For example, users can interact with the library system by the graphical user interface (GUI). The printer interacts via TCP/IP with the library system. Interaction between external systems and persons can also be modelled, e.g., the manager and an employee can communicate via speech.

3) SD and ID: Elicitation and specification of processes at the domain (e.g., business processes) is an important aspect at requirements engineering. In our approach, these processes can

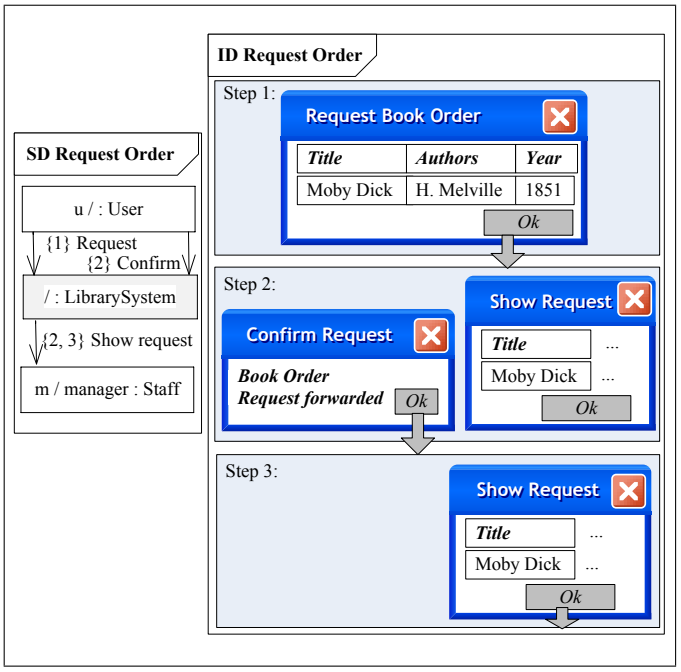


Figure 5. Scenario - Request Book Order

be described by a SD in combination with an ID. SD is based on scenario-based UML Communication Diagrams [11] and describes representative scenarios at the domain, which have to be supported by the system under development. The most important elements of an SD are lifelines and interactions. Lifelines represent instances of systems, persons and entities of the domain structure. The interactions are described by messages of the UML communication diagram and represent the interaction between the instances during a specific scenario. They are visualized by an arrow between two lifelines. The SD *Request Order* and SD *Process Request* describe scenarios in the library domain (see Fig. 5 and Fig. 6).

The SD *Request Order* describes the interactions between a user, the library system and the manager in the case of a request for a book order. An ID visualizes and describes one SD in more detail. The ID describes a set of scenario steps. The scenario *Request Order* has, for instance, the scenario steps *1*, *2* and *3*. Every interaction of an SD can be active at a set of scenario steps. The interaction *request* of the SD *Request Order* of the user with the system is, for instance, active at scenario step *1* and is labeled with this number. An active interaction is described by an interaction mockup in the corresponding ID, which is visualized by a dialog. The interaction *request* is, for instance, visualized by the interaction mockup *Request Book Order* in scenario step *1*. The visualization of the ID is suitable for the validation of scenarios by stakeholders, since the used interaction mockups give a representative view on the exchanged data. In a transition to a next scenario step, interactions can be activated and deactivated. The begin of a scenario, the end of a scenario, and a completion of an interaction can be the trigger of a transition. The completion of the interaction *Request* visualized by the

interaction mockup *Request Book Order* is, for instance, the trigger of the transition to scenario step *2*.

Several interactions can be active concurrently in one scenario step [25]. In this case, one scenario step of the ID contains two interaction mockups. The scenario step *2* contains, for instance, the interaction mockups *Confirm Request* and *Show Request*. These mockups are visualizing the interactions *Confirm* and *Show Request* of the system to the user resp. the manager. Both interactions are labeled with the number *2* of the corresponding scenario step. In the case of concurrency, an interaction might be active at several scenario steps. Consequently, several sequenced scenario steps can contain the same interaction mockup [25]. The interaction *Show Request* is active in the scenario steps *2* and *3*. Hence, the visualizing interaction mockup is contained in these steps. The interaction is labeled with the numbers *2* and *3*.

An SD and ID can also be used to describe alternative scenario sequences [25]. The SD and ID *Process Request* describes, for instance, a scenario with two alternative scenario sequences for the processing of a request for a book order (see Fig. 6). In the first alternative, the book is not ordered. In the second, alternative the book is ordered by forwarding the request to the employee and printing the book data. These alternatives are described by the scenario steps *3a* and *3b*, which can follow scenario step *2*. If the book is not ordered, the manager returns to the overview of all books after showing the books statistic and exits the overview. The interaction of exiting the overview is, for instance, described by the message *Exit* in the SD. Since this interaction is active in scenario step *3a*, it is labeled by this number. If the book is ordered, the request is forwarded to an employee and the book data is printed in parallel. The printing of the book data is, for instance, described by the interaction *Print Book*, which is active in the scenario steps *3b* and *4*.

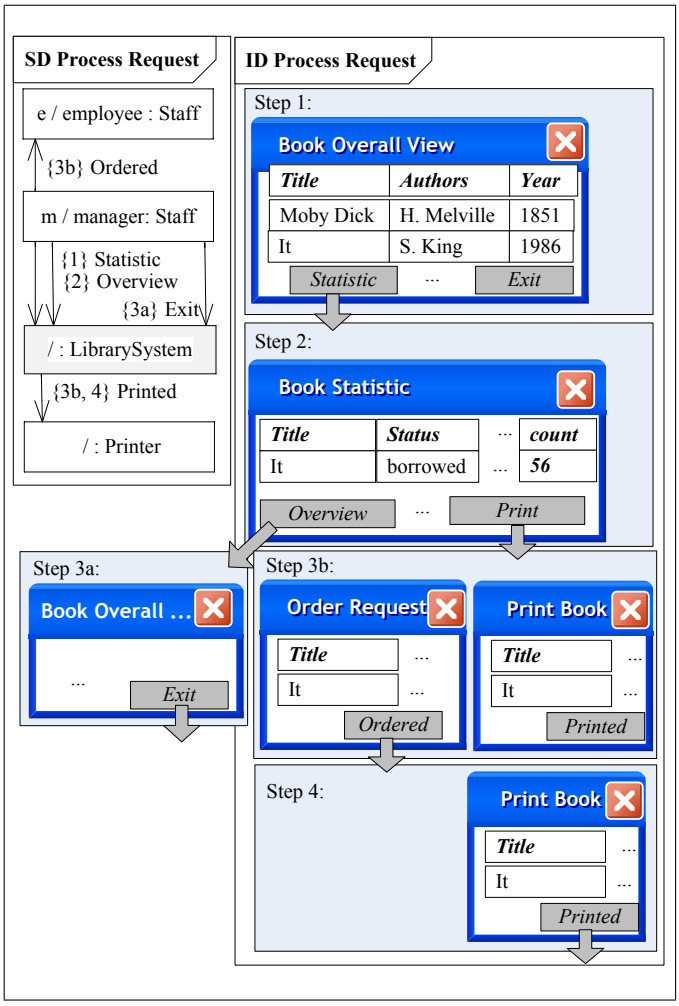


Figure 6. Scenario - Process Request

B. Architecture Description

The architecture of the library system is described by the CREATE component model. This component model allows the description of the behavior of the system under development and the resulting internal structure. The component model consists of the provided OD, ABD, DD and ASD. At the architecture design, the definition of the system boundary and the provided functions is crucial. The system boundary and the provided functions can be described by the OD of CREATE. The process of a function is described by the ABD. Data objects used in these processes are defined in the DD. A suitable structure of the system can be derived from the process descriptions. This structure is described by the ASD. The diagrams of the CREATE architecture have not to be developed in a fixed order. Typically, a small set of system functions are described by the OD, ABD and DD first. Afterwards, the ASD is derived, and finally, additional system functions are described. During the development of the architecture, consistency to the requirements is maintained continuously. In the following, all diagrams of the CREATE architecture are explained in detail by an exemplary architecture design of the library system.

1) OD: A crucial step in the architecture design is the definition of the system boundary and the privided functions of the system under development. The system boundary and the provided functions can be described precisely by the OD, which is based on the UML Use Case Diagram [11]. It describes the most abstract structure and behavior of the system and its context by the system boundary and the associated use cases, which are called *functions*. The OD of the library system describes the system with the provided functions *ShowBooksStatistic* and *PrintBookStatistic* (see Fig. 7).

The OD describes additionally all actors, which are directly involved in functions of the system under development. An actor can be a person, an external system or a hardware device. The OD of the library system describes, for instance, the actors *User*, *Staff* and *Printer*. These actors are involved in at least one function of the library system. The actor Staff is, for instance, using the function *ShowBooksStatic* and *PrintBooksStatistic*. The printer is used by the system at the function *PrintBooksStatistic*. The process for each function is described by an Architectural Behavior Diagram.

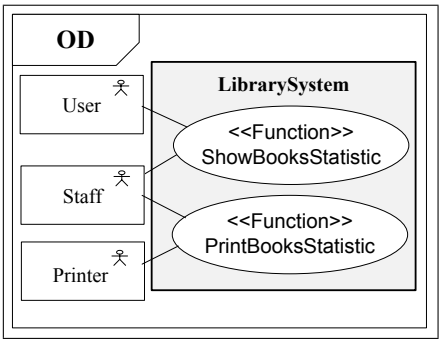


Figure 7. OD of the library system

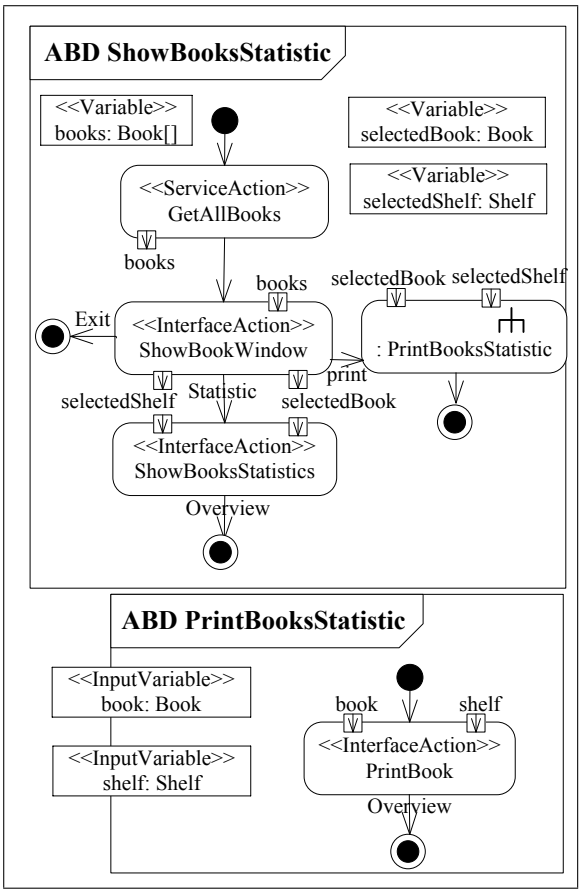


Figure 8. ABDs of the functions ShowBookStatistic and PrintBookStatistic

2) ABD: The ABD defines the behavior of the software system by describing the processes of the functions defined in OD completely. It is based on the UML Activity Diagram [11] including data flow. An activity of an ABD describes a process. In this process actions can be perfomed. The process can be described by control nodes. Control nodes allow, among others, a description of parallel and alternative execution sequences [11]. The function *ShowBooksStatistic* is, for instance, described by the ABD *ShowBookStatistic* (see Fig. 8).

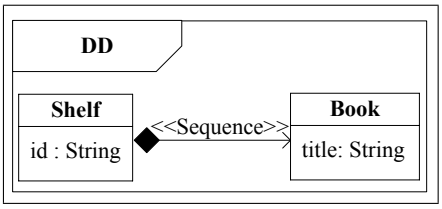


Figure 9. DD of the library system

Within the ABD, different action types like *InterfaceAction* and *ServiceAction* are used. An *InterfaceAction* describes an interaction of the system with its environment. The action *ShowBookWindow* describes, for instance, an interaction with the user by showing a dialog (see Fig. 8). A *ServiceAction* is performed by the system (e.g., a database call). The action *GetAllBooks* reads all books from the data base. An ABD supports the usage of call behavior actions [11], which describe the execution of a called activity within the execution of another activity. The action *:PrintBooksStatistic* describes the call of the ABD *PrintBooksStatistic*.

The ABD uses variables for the description of the data flow of processes [25]. In the description of the function *ShowBookStatistic*, the variables *books*, *selectedBook* and *selectedShelf* are used. Variables can have different data types and can hold a set of objects. The variable *books* holds, for instance, a set of objects of the type *Book* (see Fig. 8). Reading and writing of variables by actions can be described by input pins and output pins. The action *GetAllBooks* writes, for instance, all selected book objects into the variable *books*. This variable is read by the action *ShowBookWindow*, which shows, among others, an overview about the selected books.

At a call of another ABD parameters can be passed. For the passing of parameters a copy semantics is used. The action *:PrintBooksStatistic* of the ABD *ShowBooksStatistic* describes, for instance, a call of the ABD *PrintBooksStatistic*. The ABD *PrintBooksStatistic* has the input variables *book* of the type *Book* and *shelf* of the Type *Shelf* (see Fig. 8). The action *:PrintBooksStatistic* has two input pins, which are referring to the variables *selectedBook* and *selectedShelf*. At a call, the objects in the referred variables are copied to the input variables of the called ABD. At a return, the objects of the output variables are copied to the referred variables of the output pins of the call behavior action.

3) DD: At a function, described by ABD, data objects can be used by the system. The DD is based on UML Class Diagrams [11] and describes the data types of each data object. For example, the DD of the library system describes a type *Book* and *Shelf*, which are types of the ABD *ShowBooksStatistic* variables (see Fig. 9).

Data types described in the DD can have attributes similar to classes in the UML class diagramm. Due to the copy semantics of the ABD and the complexity of copying object networks, attributes can only have a primitive type in the DD. Relations to other data types are only described by sequences and generalisations. A sequence is a special composite aggregation [11], which allows no cylces and no membership of one object to more than one other object. In the DD of the library system

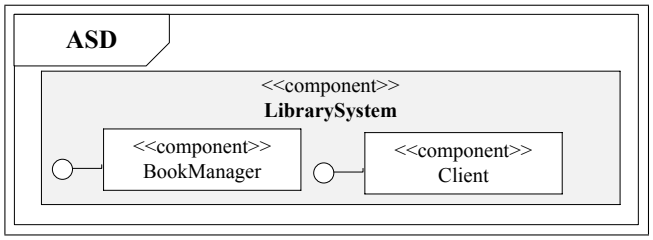


Figure 10. ASD of the library system

the data type *Shelf* has a sequence of objects of the type Book. The meaning of a generalization of data types is equal to the generalization of classes in the UML class diagram [11].

4) ASD: The ASD is based on UML Component Diagrams [11] and describes the internal components of the system under development and their offered interface as a black-box view. Subsequently, the components are further decomposed to refine their internal structure. The ASD *LibrarySystem* describes, for instance, the internal structure of *LibrarySystem* of the OD (see Fig. 10). The internal structure is derived from the actions of the ABD. Hence, each component must be associated with at least one action of an ABD. The component *LibrarySystem* is, for example, refined by a component *BookManager*, which is associated with the action *GetAllBooks* of the ABD.

V. Consistency constraints

In CREATE, consistency between requirements and architectures is maintained continuously during the architecture design. Consistency maintenance is supported by defined interrelations between scenario-based requirements models and component-based architecture models. Interrelations are defined by constraints. The definition of these constraints support the consistency maintenance, because they can easily be checked. Furthermore, they enable an automation of the consistency maintenance [24]. In this section, these interrelations are explained in detail based on the library system example described in Section IV. A summary of these interrelations is given in Fig 11. We distinguish three kinds of interrelations:

1) Interrelations within requirements (see gray dotted lines between the requirements diagrams in Fig. 11, e.g., between HRL and SD).
2) Interrelations within architecture (see gray dotted lines between the architecture diagrams in Fig. 11, e.g., between OD and ASD).
3) Interrelations between requirements and architecture (see red dotted lines between diagrams of the requirements and diagrams of the architecture in Fig. 11, e.g., between the DSD and the OD).

One interrelation between requirements and architectures is, for instance, that the system boundary of the OD represents a part of the domain structure in the DSD. In the following, the interrelations within requirements, the interrelations within architectures and the interrelations between requirements and architectures are explained completely and in detail. The interrelations are explained by the example of the requirements model and architecture model of the library system.

A. Constraints within requirements

CREATE defines strict interrelations between the requirements models. These interrelations are defined by consistency constraints. An inconsistency means a violation of these constraints. Within requirements the following consistency constraints are defined:

1) Every lifeline in a SD must have a corresponding part with the same type in the DSD.
2) If an interaction in a SD takes place between two lifelines, a connector has to exist between the corresponding parts in the DSD.
3) In a scenario described by an SD the number of instances have to comply with the multiplicity of the corresponding parts in the DSD.
4) Every interaction of the SD is described by exactly one interaction mockup of the ID and every interaction mockup describes exactly one interaction.
5) Every text-based requirement on the system of the HRL is described by at least one SD.
6) Marked subjects in the HRL are described in the DSD.
7) Every marked subject in the HRL must be used in the SD, which describes this text-based requirement.

The requirements model of the library system example comply with all of these constraints. For instance, the lifeline *manager* of the SD *Process Request* has a corresponding part with the same type *Staff* in the DSD (see Fig. 11). A connection exists between the part *manager* and the part of the library system. In this way, the interaction *Statistic* between the manager and the library system complies to the constraint *2*. Further, exactly one library system is used in the SD *ProcessRequest*. This complies with the constraint *3*. The constraint *3* is fulfilled for every scenario of the library system described by an SD.

In the following, typical changes of requirements are introduced in order to show the inconsistency detection and solving of the CREATE approach. A typical change during the development of a software system is the addition of a new scenario. In a new scenario of the library system example, the employee has a look at the book statistics. This scenario is described by a new ID *ShowBookStatistic* and a new SD *ShowBookStatistic* (see Fig. 12). Furthermore, the text-based requirement *1* in the HRL is changed, which states that employees and managers need to have a look at book statistics. During the development of this scenario, it is discovered that employees should not be able to see all book information like the manager. Hence, the lifeline of the employee in the SD is not of the type *Staff*, but of the type *Employee* (see Fig. 12). Employee is not defined in the initial DSD (see Fig. 4). In consequence, the consistency constraint *1* is violated. This inconsistency is fixed by introducing the types *Employee* and *Manager* in the DSD and assigning these types to the parts *employee* resp. *manager*. This leads to another inconsistency with the SD *Process Request*. The lifeline *manager* and *employee* are of the type *Staff* in the initial version of the SD. But this type is not defined in the DSD anymore. This inconsistency is fixed by changing the type of the part *employee* from *Staff* to *Employee* and the type of the *manager* from *Staff* to *Manager*. Furthermore,

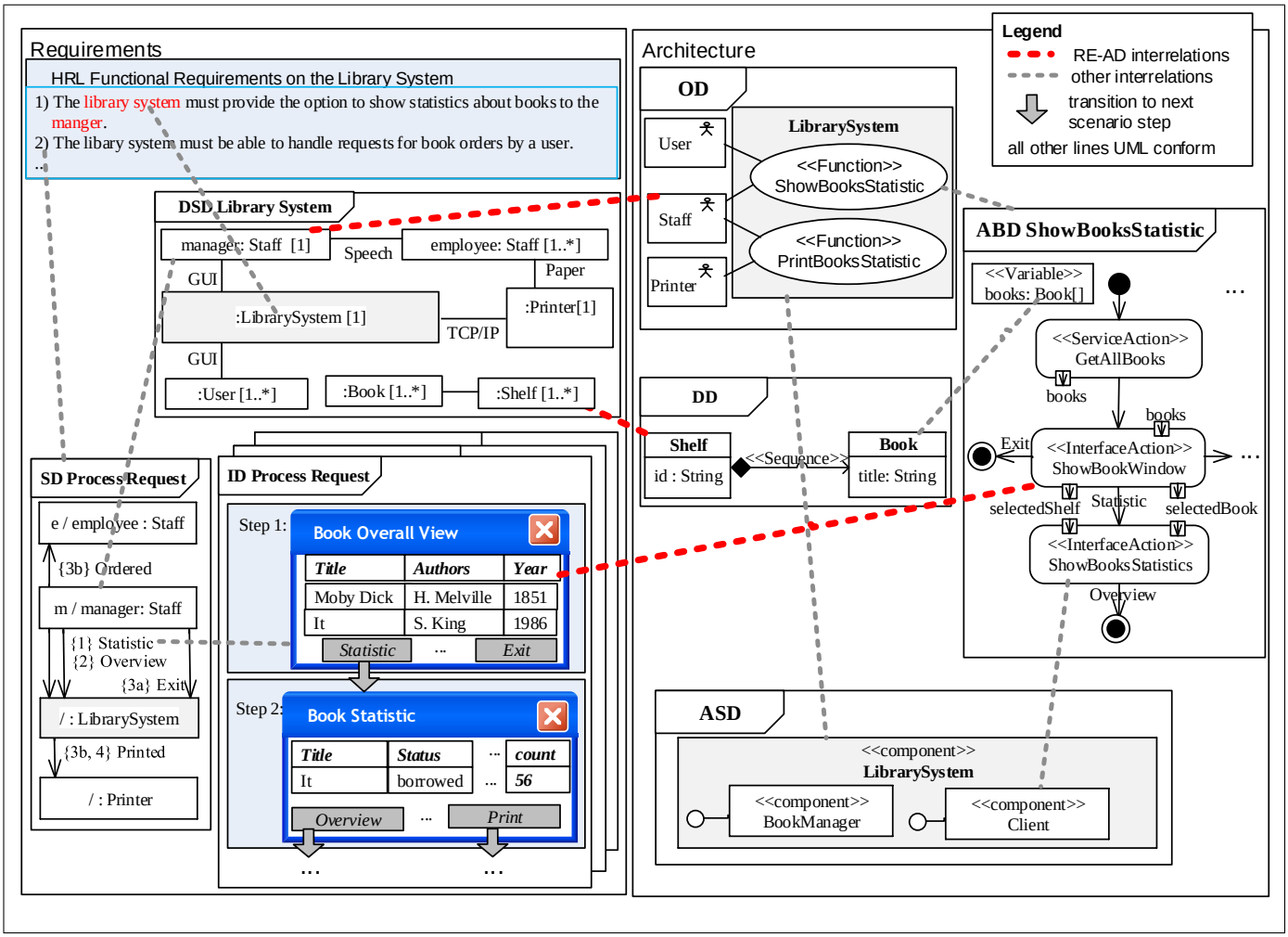


Figure 11. Requirements models, architecture models and interrelations

the consistency constraint *2* is initially violated by introducing the SD *ShowBookStatistic*. The employee interacts with the library system, but a connector is not existing between the corresponding parts of the DSD. To solve this inconsistency a new connector has to be added between the parts for the employees and the library system.

B. Constraints within architectures

In CREATE, strict interrelations between architectures diagrams are defined. These interrelations are also defined by consistency constraints. A violation of a consistency constraint points out an inconsistency. Within architectures the following consistency constraints are defined:

1) Every system function has to be described by one ABD.
2) Every type of a variable in an ABD has to be primitive or must be defined in the DD.
3) Every input pin and output pin of a call behavior action must have a corresponding input variable resp. output variable of the called ABD and vice versa.
4) The types of the pins of a call behavior action must match the types of the corresponding variables of the called ABD.
5) The system boundary of the OD has to be decomposed by the ASD.
6) Every action described in one ABD has to be realized by exactly one component of the ASD.
7) Every component of the ASD has to realize at least one action of one ABD.
8) An actor has to be involved in a system function.

The architecture model of the library system example comply with all of these constraints. For instance, the function *ShowBookStatistic* of the OD is described by the ABD *ShowBookStatistic* (see Fig. 11). The type *Book* is defined in the DD and the input pin of the call behavior action *:PrintBookStatistic* referring to the variable *selectedBook* is, for instance, corresponding to the input variable book of the called ABD *PrintBookStatistic*. Further, the library system is decomposed in the ASD. The action *ShowBookStatistic* is, for instance, realized by the component *Client* in the DSD.

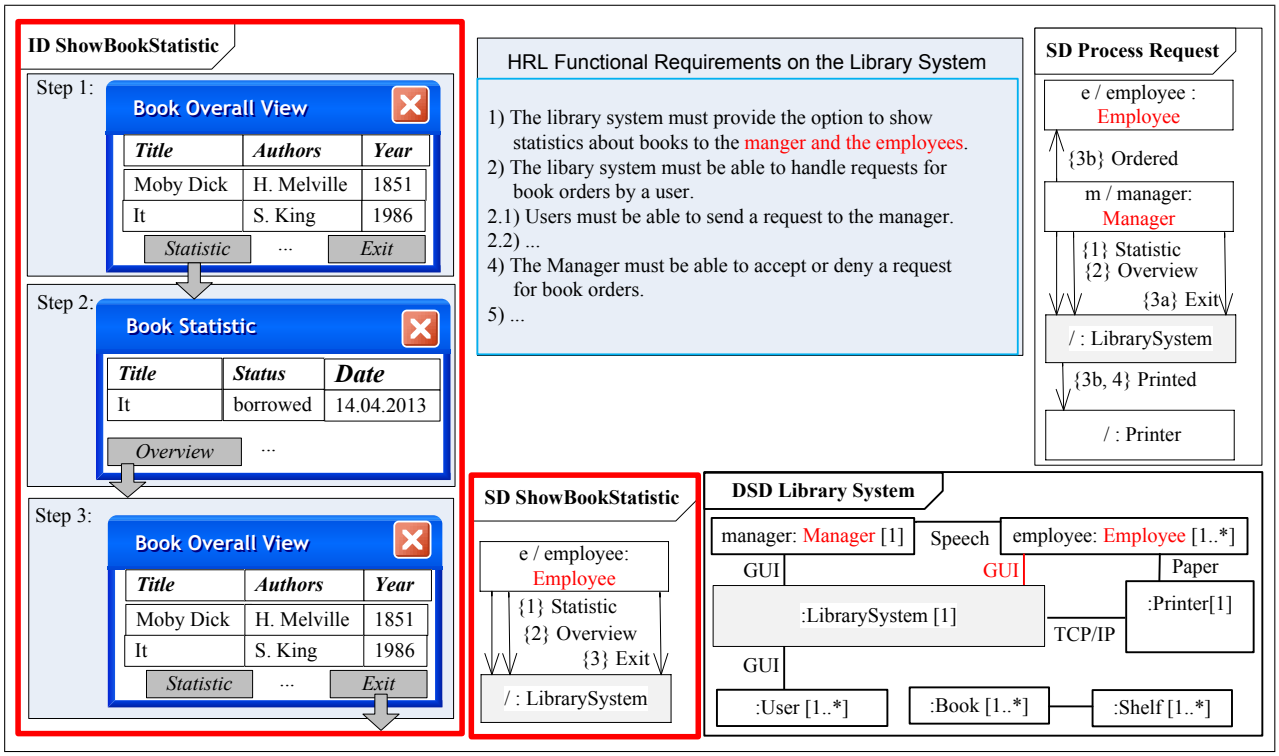


Figure 12. Change within requirements

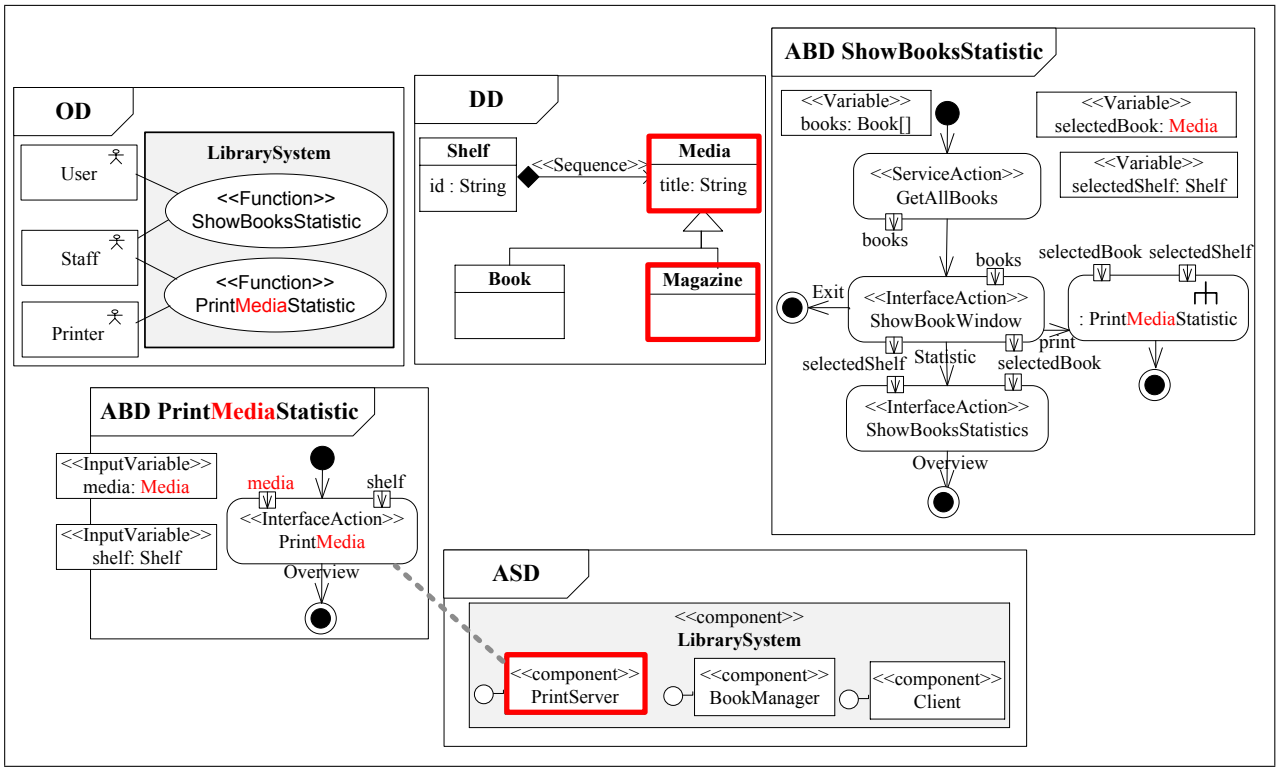


Figure 13. Change within architecture

In the following, typical changes of architectures are introduced in order to show the inconsistency detection and solving of the CREATE approach. A typical change during the development of an architecture is that the function can be reused in another context. In the library system example, the function *PrintBookStatistic* can only print the statistics of books. This function is changed to be able to print all data about media in general. Especially the library system should be able to handle also magazines. The ABD *PrintBookStatistic* is changed to *PrintMediaStatistic* (see Fig. 13). Due to this change, the type of the input variable is changed from *Book* to *Media*. The type Media is not defined in the initial DD (see Fig. 9). In consequence, the consistency constraint *2* is violated. This inconsistency is fixed by introducing this type in the DD. Further, the condition *4* is violated. The input pin of the call behavior action *:PrintBookStatistic* is, initially, referring to the variable *selectedBook* of the type *Book* (see Fig. 10) and the corresponding input variable of the called ABD *PrintBookStatistic* is of the type *Media* (see Fig. 13). One way to fix this inconsistency is to change the type of the passed variable *selectedBook* to *Media*. In order to allow the printing of book data by the function *PrintMediaStatistic*, the generalization between the type *Book* and the new type *Media* in the DD is introduced. Further, a new type *Magazin* is added, which is also generalized by the type *Media*. The action *PrintMedia* of the ABD *PrintMedaStatistic* is realized by a new component *PrintServer* of the ASD.

C. Constraints between requirements and architectures

Finally, CREATE defines strict interrelations between requirements and architectures. These interrelations are again defined by consistency constraints. An inconsistency means a violation of these constraints. Within architectures the following consistency constraints are defined:

1) The system boundary of the OD has to represent one type of the parts in the DSD.
2) Every actor of the OD has to represent one type of the parts in the DSD.
3) The existence of a type in DSD whose part is directly connected with the part of the system to build implies the existence of a corresponding actor in the OD.
4) The existence of an entity in the DD implies the existence of a corresponding type in the DSD.
5) The existence of a relation between two entities in the DD implies the existence of a connection between parts of the corresponding types in the DSD.
6) Every interaction mockup in the ID visualizing an interaction with the system must be realized by exactly one *InterfaceAction* in an ABD.
7) A system function of the OD realizes a set of interactions described in at least one SD.
8) A type in the DSD can either be represented by an actor of the OD or by an entity of the DD.

The initial requirements model and the initial architecture model of the library system example comply with all of these constraints. For instance, the system boundary of the library system in OD represents the type *LibrarySystem* of the DSD (see Fig. 11). The actor *Manager* represents, for instance, the type *Staff*. In this way, the connection between the parts of the library system and the part manager is valid. Between the entities *Book* and *Shelf* exists, for instance, a sequence relation and between the parts of the corresponding types in the DSD exists a connection. Every interaction mockup described in the IDs is realized by one interface action. For instance, the interaction mockup *Book Overall View* is realized by the interface action *ShowBookWindow* (see Fig. 11).

In the following, the changes of the requirements and the architecture of the library system in the previous sections are considered in order to show the inconsistency detection and solving of the CREATE approach. Due to the changes in the requirements specification the type *Employee* is introduced in the DSD. The part *employee* of this type is connected with the system in the DSD. Since no corresponding actor is described in the OD, the consistency constraint *2* is violated (see Fig. 14). The new data types *Media* and *Magazin* are added to the architecture during the further development of the architecture design according to the previous sections. In the DSD no part of these types is defined (see Fig. 14). As a result of this, the consistency constraint *4* is violated. Further, in our example the new interaction mockup *Book Statistic* is introduced in the new scenario described in the ID *ShowBookStatistic*. This interaction mockup is not realized by an interface action of an ABD in the architecture (see Fig. 14). Hence, the consistency constraint *6* is violated.

To correct these inconsistencies, a few further changes have to be made. It is necessary to add the entities *Media* and *Magazine* to the DSD. After this consistency condition *4* holds again (see Fig. 15). To comply with the consistency constraint *2*, a new actor for the employee has to be introduced into the OD (see Fig. 15). Finally, a mapping from the added interaction mockup to an interface action is missing. One could map the new interaction mockup to an existing interface action or extend the ABD by a new interface action. By extending the ABD by the interface action *EmployeeStats* the interaction mockup can be mapped on it (see Fig. 15). In this way, every interaction mockup is realized by one interface action and the model complies with the constraint *6*.

As shown above, the defined consistency conditions help at the consistency maintenance. The conditions can easily be checked. In this way, inconsistencies can be detected and solved. Furthermore, these consistency conditions enable an automatic support of the consistency maintenance [24]. An automatic consistency maintenance can, for instance, be realized by permitting changes to a next version not until all inconsistencies are solved.

VI. Evaluation

The development of CREATE took place at research projects in cooperation with a public institution over a period of four years. At these research projects, we gave advice and supported to system development projects in order to test our results in practice. The goal of the overall approach is to support consistency maintenance of requirements and architectures in early development phases. In these early phases, requirements

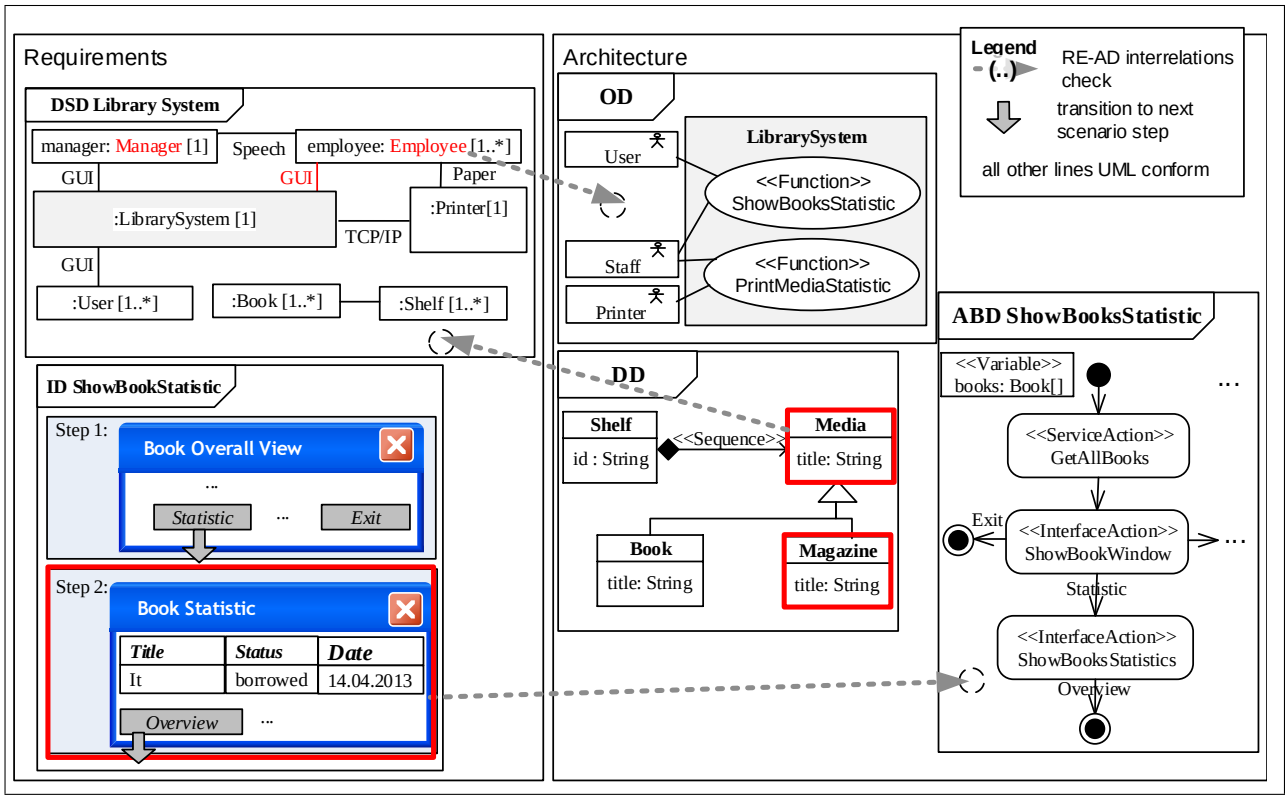


Figure 14. Changes at the requirements and architecture model

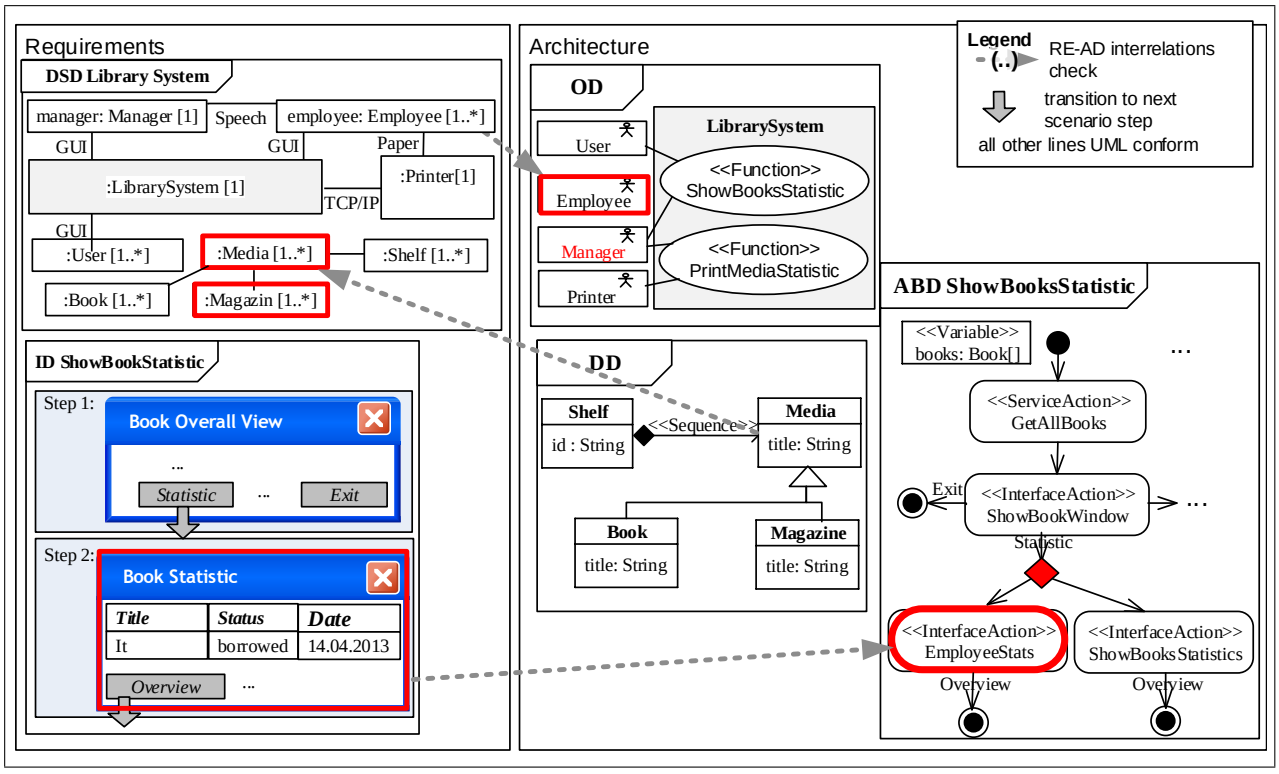


Figure 15. Changes to solve the inconsistencies

and architecture decisions are frequently changing. Further, feedback of stakeholders is crucial since the requirements on the system have still to be understood more clearly. The goal of the evaluation was to test the usability and the inconsistency prevention of our approach.

In a first step, we developed the component-based architecture model for a precise description of the architecture. For reconcilement with stakeholders we developed a prototype generator, which is able to interpret the developed models. The stakeholders should validate the architecture and the consistency to their requirements with the aid of the prototypes. This approach was tested at a system development project over a period of one year. The subject of this project was a communication system. At this project, a model of the complete system was developed comprising 20 system functions, 253 activity nodes and 35 data types. The models were developed in the sparx enterprise architect tool [26]. Conclusive, it revealed that the usability of the approach has to be improved. The number of possible states described by the component-based architecture leads to less comprehensibility to stakeholders. They were not able to agree to the developed specifications. Furthermore, the impact of changes was not readily understandable to stakeholders. Consequently, the consistency maintenance of requirements and architectures could not be supported by this approach.

Based on the results of this practice test we extended the approach by scenario-based models. This extended co-modeling approach was the model-based approach CREATE, which is described in detail in this paper. It was tested in practice at a further system development project with a similar subject over a period of one year. Besides the sparx enterprise architect we used balsamiq mockups tool for the description of the interaction mockups. In this period, the usability was significantly better. The required resource demand for creating and maintaining the models was still high with round about 4 person month. But stakeholders were able to agree to the visualized and scenario-based requirements. Furthermore, they were able to give helpful feedback, which leads to a big number of changes. We measured at three milestones the number of changes, the detected errors and especially remaining inconsistencies. Between these milestones we documented 500 changes and 67 errors. 8 of these errors were inconsistencies. The rate of inconsistencies to changes is low. For an indication, at a study described in [23], change data of requirements documents are analyzed. In this study, 88 changes, 79 errors, and 16 inconsistencies were detected. Furthermore, the resource demand for consistence maintenance was low. Only 8 inconsistencies had to be resolved. Parts of the the consistence checks could even be checked automatically.

VII. Conclusion and Future Work

The fundamental problem addressed in this paper was the development of inconsistencies between requirements and architectures at the advanced approaches for the iterative evolution. In this paper, the model-based approach CREATE [1] was described in all details, which supports the iterative evolution of requirements and architectures. The approach uses a scenario-based model for a precise description of requirements and a component-based model for the description of architectures. The architecture of CREATE describes precisely how requirements are fulfilled by the system under development. Requirements and architectural decisions lead frequently to inconsistencies between requirements and architectures. CREATE supports the consistency maintenance during the development of requirements and architectures by defined interrelations between scenario-based requirements models and component-based architecture models. The definition of these constraints support the consistency maintenance, because they can easily be checked. Furthermore, they enable an automation of the consistency maintenance [24]. This addresses the important concern of the scalability of the method when it is applied in complex industrial systems. During the development of such a system a large variety of requirements and architectural decisions have to be made. Since the consistence maintenance can be automated the approaches scales well with the size of the project.

A frequently stated argument is the entailment of high costs for the development of precise requirements and architecture models at a software project. This can be countered by the fact that an incorrect consideration of requirements not uncommonly leads to complete project failures. Thus, maintaining the consistency at the iterative evolution of requirements and architectures is important. Supporting this task by models enabling an automatic consistency maintenance reduces the risk of a project failure and costs for consistency maintenance. Furthermore, the developed models can be reused for automatic generation of code, test cases and documents like, for instance, requirements specifications. Nevertheless, the usage of formal models at a development project should, among others, be made conditional on the size of the project. At the beginning of a development project, the advantages and disadvantages of using formal models have to be weighed.

As future work, a further evaluation is planned to compare the effectivity of CREATE to other model-based approaches for requirements and architectures. Furthermore, it is planned to develop a tool for the automatic consistency maintenance.

References

[1] M. Ibe, M. Vogel, B. Schindler, and A. Rausch, "CREATE: A co-modeling approach for scenario-based requierements and component-based architectures," in Proceedings of the International Conference on Software Engineering Advances (ICSEA), IARIA XPS Press, 2013, pp. 220-227.

[2] B. Nuseibeh, "Weaving together requirements and architectures," IEEE Computer Society Press, vol. 34, March 2001, pp. 115–117.

[3] W. W. Royce, "Managing the development of large software systems: concepts and techniques," in Proceedings of the 9th International Conference on Software Engineering, IEEE Computer Society Press, 1970, pp. 1–9.

[4] B.W. Böhm, "A spiral model of software development and enhancement," IEEE Computer Society Press, vol. 21, May 1988, pp. 61–72.

[5] P. Grünbacher, A. Egyed, E. Egyed, and N. Medvidovic, "Reconciling software requirements and architectures with intermediate models," in Software and Systems Modeling. Springer, 2003, pp. 202–211.

[6] R. Ferrari and N. H. Madhavji, "The impact of requirements knowledge and experience on software architecting: an empirical study," in Working IEEE/IFIP Conference on Software Architecture, 2007, pp. 44–54.

[7] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in Proceedings of the Conference on The Future of Software Engineering, ACM Press, 2000, pp. 35–46.

[8] C. Knieke and U. Goltz, "An executable semantics for UML 2 activity diagrams," in Proceedings of the International Workshop on Formalization of Modeling Languages, ACM Press, 2010, pp. 3:1–3:5.

[9] M. Elkoutbi, "Automated prototyping of user interfaces based on UML scenarios," in Journal of Automated Software Engineering, vol. 13, Kluwer Academic Publishers, 2006, pp. 5–40.

[10] K. Schneider, "Generating fast feedback in requirements elicitation," in Proceedings of the 13th international working conference on Requirements engineering: foundation for software quality, Springer-Verlag, 2007, pp. 160–174.

[11] OMG, "UML, version 2.2. OMG specification superstructure and infrastructure," 2009.

[12] A. G. Kleppe, J. Warmer, and W. Bast, "MDA explained: the model driven architecture: practice and promise," Addison-Wesley Longman Publishing Co. Inc., 2007

[13] C. Atkinson, J. Bayer, and D. Muthig, "Component-based product line development: the KobrA approach," in Software Product Line Conference, Denver, Kluwer Academic Publishers, 2000, pp. 289-309.

[14] R. Chitchyan, M. Pinto, A. Rashid, and L. Fuentes, "COMPASS: composition-centric mapping of aspectual requirements to architecture," in Transactions on AspectOriented Software Development, 2007, pp. 3–53.

[15] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-based analysis of software architecture," in IEEE Software, vol. 13, IEEE Computer Society Press, Nov. 1996, pp. 47–55.

[16] H. Liang, J. Dingel, and Z. Diskin, "A comparative survey of scenario-based to state-based model synthesis approaches," in Proceedings of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools, ACM Press, 2006, pp. 5–12.

[17] Y. Bontemps, P. Schobbens, and C. Löding, "Synthesis of open reactive systems from scenario-based specifications," in Proceedings of Application of Concurrency to System Design, 2003, pp. 41–50.

[18] V. Garousi, L. Briand, C. Lionel, and Y. Labiche, "Control flow analysis of UML 2.0 sequence diagrams," in Model Driven Architecture Foundations and Applications, 2005, pp. 160–174.

[19] C. Szyperski, "Component software: beyond object-oriented programming," Addison-Wesley Longman Publishing Co. Inc., 2002.

[20] A. Rausch, R. Reussner, R. Mirandola, and F. Plasil, "The common component modeling example: comparing software component models," ser. Springer Lecture Notes in Computer Science, vol. 5153, 2008.

[21] A. Egyed, E. Letier, and A. Finkelstein, "Fixing inconsistencies in UML design models," in Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008, pp. 99–108.

[22] Y. Aoki, H. Okuda, S. Matsuura, and S. Ogata, "Data lifecycle verification method for requirements specifications using a model checking technique," in Proceedings of the International Conference on Software Engineering Advances (ICSEA), IARIA XPS Press, 2013, pp. 194–200.

[23] V. R. Basili and D. M. Weiss, "Evaluation of a software requirements document by analysis of change data," in Proceedings of the 5th International Conference on Software Engineering, IEEE Press, 1981, pp. 314–323.

[24] B. Schindler, and A. Rausch, "Automatic consistence maintenance of requirements and architectures," in Proceedings of the IASTED International Conference on Software Engineering, ACTA Press, 2014, pp. 15–22.

[25] B. Schindler, "Konsistenzsicherung von Anforderungen und Architekturen," Technische Universität Clausthal, 2014.

[26] D. Steinpichler, "Project management with UML and Enterprise Architect," SparxSystems Eigenverlag, 2011

When May the Accuracy of Expert Estimation Be Improved by Using Historical Data?

Gabriela Robiolo
Universidad Austral
Av. Juan de Garay 125
Buenos Aires, Argentina
grobiolo@austral.edu.ar

Silvana Santos
Universidad Nacional de La Plata
Calle 50 y 20
La Plata, Argentina
silvanasantos@gmail.com

Bibiana Rossi
Univ. Argentina de la Empresa
Lima 717
Buenos Aires, Argentina
birossi@uade.edu.ar

***Abstract*— As expert estimation is the estimation strategy most frequently applied to software projects today, it is important to focus the research on effort estimation methods on it. This is the estimation method used in Agile contexts so we are interested in deeply understanding the value of historical data in this context, in particular when the project domains and the technological environments are new to the team, and when the teams -with little experience in Agile contexts- have recently been created. We designed an empirical study in order to find out when the accuracy of expert estimation made in a context of agile software development may be improved by using historical data. Our empirical study has shown that the use of historical data may improve the intuitive expert estimation method under the following circumstances: when the work experience, the experience in the technologies to be used to develop the application, and the experience in a given domain is low, as well as when the team velocity is unknown.**

Keywords—Expert, Expert Estimation, Effort Estimation, Empirical Study, Historical Data.

I. INTRODUCTION

Expert estimation is the estimation strategy which is most frequently applied today to estimate the effort involved in the development of software projects. However, the estimations thus obtained are far from being as accurate as desirable. If we expect to improve estimation accuracy, further research should be carried out in order to understand how the estimation process works. At present, we are particularly interested in expert estimation in Agile contexts, so we would like to learn if historical data may bring any improvement to expert estimation. To pursue this objective, we are now extending a paper we wrote last year [1], which was presented at ICSEA 2013, in order to add more evidence in favor of using expert estimation. This new paper will also confirm the evidence reported by other authors in [2].

Having decided on our goal, we found out that the compilation of information about cost estimation made by Jørgensen and Shepperd [3] in 2007 was extremely valuable, since they systematically reviewed papers on cost estimation studies and they provided recommendations for future research. They found out that there are few researchers working in this field and that there is no adequate framework to develop high quality research projects that may lead to conclusive evidence.

Consequently, they suggested the following improvements in the field of research:

- *Deepen the study of the basic aspects of software estimation.* Jørgensen and Shepperd focused on two basic aspects: the evaluation of the accuracy of an estimation method and the appropriate selection of an estimation method.
- *Widen the research on the current, most commonly used estimation methods in the software industry.* The leading estimation method today is that based on expert opinion (ranging from analogies to experiences and intuition), but research on expert estimation is still scarce.
- *Perform studies which support the estimation method based on expert judgment, instead of replacing it with other estimation methods.* Given the fact that expert judgment is the most widely used method in the software industry today, it would be convenient to improve such judgment by supporting it with the use of formal estimation methods.
- *Apply cost estimation methods to real situations.* There are few studies in which the estimation methods are evaluated in real situations, since most of such methods are applied to laboratory, non realistic contexts.

In Agile contexts, in particular, there is another critical aspect to be dealt with: not knowing the velocity at which the developing team works. Actually, Cohn [4] suggested that one of the challenges when planning a release is estimating the velocity of the team. He mentioned three possible ways to estimate velocity. Firstly, estimators may use historical averages, if available. However, before using historical averages, they should consider whether there have been significant changes in the team, the nature of the present project, the technology to be used, and so on. Secondly, estimators may choose to delay estimating velocity until they have run a few iterations. Cohn thinks that this is usually the best option. Thirdly, estimators may forecast velocity by breaking a few stories into tasks and calculating how many stories will fit into the iteration.

Bearing in mind the present working conditions, as described in the two previous paragraphs, and in order to deepen our knowledge about expert estimation, as

recommended by Jørgensen and Shepperd [3], we decided to research on the importance of historical data when performing expert estimations in agile contexts in which the project domains and the technological environments are new to the team, and the teams -with little experience in Agile contexts- have recently been created, so the team velocity is unknown.

It is important to note that this study does not take into consideration the effect of non-functional user requirements on effort; hence it will not address the estimation of effort which is necessary to satisfy non-functional requirements.

In this scenario, we have tried to answer the following research question: *when may the accuracy of an expert estimation made in a context of agile software development be improved by using historical data?* The results we obtained through our empirical study, both those in [1] and the ones in this new paper, in which we have included a different way of applying one of the estimation methods reported in [1] and more detailed results, have led us to conclude that historical data may improve the accuracy of an intuitive estimation made by an expert when the estimator has limited experience in the job to be performed, the technologies to be used and the domain to be dealt with, and when the team velocity is unknown.

In the following Section, we will investigate related work to see if there is any other evidence of improvement in expert estimation accuracy when using historical data. In Section III, we will introduce three estimation methods: Expert Estimation (ExE), Analogy-Based Method (AbM), and Historical Productivity (HP). In Section IV, we will describe an empirical study and in Section V we will analyze the results obtained. Moreover, the threats to validity will be discussed in Section VI and finally, in Section VII, we will draw conclusions about the evidence which shows the benefits of using historical data.

II. RELATED WORK

Apparently, this has been the first article to have been written about whether using historical data in an agile context improves expert estimation. However, if we consider expert estimation in general, there are some authors that have already reported evidence about the importance of the developers' level of maturity when evaluating the accuracy of estimations, which is in line with the conclusions of our study. For example, SCRUM pioneers believe it is acceptable to have an average error rate of 20% in their results when using the Planning Poker estimation technique, but they have admitted that this percentage depends on the level of maturity of the developers [5]. Another study [6] agrees with this statement, as it indicates that the optimism bias which is caused by the group discussion diminishes, or even disappears, as the expertise of the people involved in the group estimation process increases.

On the other hand, another study [7] has already examined the impact of the lack of experience of the estimators in the domain problem, as well as that in the technologies used in a software development project. In fact, what was studied was the accuracy with which the effort of a given task was estimated. Such estimation was performed by a single expert by comparing the estimated and the actual efforts. The reason for researching on this aspect is that sometimes organizations do not have in their staff experts that have relevant prior experience in some business or technology related aspect of the project they are working on. This research investigates the impact of such incomplete expertise on the reliability of estimates.

It is important to note that Jorgensen [2] has both defined a list of twelve "best practices", that is to say, empirically validated expert estimation principles, and also suggested how to implement these guidelines in organizations. One of the best practices he proposed is to use documented data from previous development tasks and another one is to employ estimation experts with a relevant domain background and good estimation records. Actually, our article goes in the same direction; we have focused on historical data and analyzed the impact of the difference in experts' skills.

An aspect that should be taken into account when performing expert estimations is excessive optimism, as it is one of the negative effects that influences the most when a software project fails. Jørgensen and Halkjelsvik [8] have made a discovery that seems to be important to understand what may be leading estimators to excessive optimism: the format used to word the question that asks about effort estimation. The usual way to ask about effort estimation would be: "How many hours will be used to complete task X?". However, there are people who would say: "How many tasks could be completed in Y hours?". Theoretically, the same results should be obtained by using any of the two formats. Nevertheless, according to Jørgensen and Gruschke [9], when the second option is used, the estimations which are thus obtained are much lower than those obtained when the traditional format is used, that is to say, the time to fulfill a task will be shorter, and consequently, the estimation will be much more optimistic. Thus, in our study, the expert estimations were made using the usual question. In fact, the final recommendation of this study is that the traditional format should always be used, as this does not contain any deviation imposed by the clients who ask the developers for more than they can pay for.

Besides the papers mentioned above, Jorgensen has written several studies that include other aspects that may affect expert estimations. Although such aspects were not taken into account in this study, we believe they may enrich our conclusions. These aspects are:

a. high degree of *inconsistency and an improper weighting of variables* [2], he believes that if these negative aspects could be reduced, the accuracy of the estimations would be much better.

b. the level of interdependence (focusing on relations, social context and interconnections) introduces a deviation in the estimation process [9], according to Jørgensen, the estimations performed by software developers are also

affected by human relationships. Besides, he points out that such deviations take place under every circumstance.

III. ESTIMATION METHODS

This section will describe the three estimation methods used in our empirical study: ExE, AbM and HP. However, before doing so, it is important to focus on the definition of certain expressions used to define such methods. For example, when defining expert, Jorgensen [2] used a broad definition of the phrase, as he included estimation strategies that ranged from unaided intuition ("gut feeling") to expert judgment supported by historical data, process guidelines, and checklists ("structured estimation"). In his view, for an estimation strategy to be included under the expert estimation category, it had to meet the following conditions: firstly, the estimation work must be conducted by a person who is considered an expert in the task, and secondly, a significant part of the estimation process must be based on a non-explicit and non-recoverable reasoning process, i.e., "intuition". In our study, however, a narrower definition of the concept of expert was used: that which refers only to intuition. This way, we made a difference between intuitive ExE, and the methods that involve the use of historical data: AbM and HP. It is important to note that in our study, when we used Planning Poker –an ExE method-, no historical data was taken into account.

To further clarify the terms used, we must say that by AbM we meant the estimation performed by an expert, who is aided by a database containing information about finished projects [11]. As regards HP, which is another way of using historical data, it is worth mentioning that in our empirical study we focused on the size characteristic of the products, as suggested by one of the authors that inspired this article [10].

A. Expert Estimation Method (ExE)

When estimating the effort of a software development task, an expert estimation may be obtained either by a single expert, whose intuitive prediction will be considered an expert judgment, or by a group of experts, whose estimation will combine several experts' judgments.

A very frequently used way to obtain group expert judgment is called Planning Poker -a technique that combines expert opinion, analogy, and disaggregation-, which is a variation of the Delphi method. Planning Poker is based on the consensus that is reached by the group of experts who are performing an estimation; in fact, it is considered a manageable approach that produces fast and reliable estimations [4][11][12]. This method was first described by James Greening [14] and it was then popularized by Mike Cohn through his book "Agile Estimating and Planning" [4]. It is mainly used in agile software development, especially in Extreme Programming [13]. To apply Planning Poker, the estimation team should be made up of, ideally, all the developers within the team, that is, programmers, testers, analysts, designers, DBAs, etc. It is important to bear in mind that, as this will happen in Agile contexts, the teams will not exceed ten people [4]. In fact, Planning Poker becomes especially useful when estimations are taking too long and part of the team is not willing to get involved in the estimation process [14]. The basic steps of this technique, according to how Grenning described it, are:

"The client reads a story and there is a discussion in which the story is presented as necessary. Then, each programmer writes his estimation on a card, without discussing his estimation with anyone else. Once every programmer has written down his estimation, all the cards are flipped over. If all estimates are equal, there is no need for discussion; the estimate is registered and the next story is dealt with. If the estimates are different, the team members will discuss their estimates and try to come to an agreement" [14].

Mike Cohn further developed this technique. He added a pack of cards especially designed to apply it. Each pack has to be prepared before the Planning Poker meeting and it will contain cards with numbers written on them. Such numbers should be big enough to be read from the other side of a table. Those numbers represent a valid estimation, such as 0, 1, 2, 3, 5, 8, 13, 20, 40, and 100. There is a raison d'être for such an estimation scale: there are studies which have demonstrated that we are better at estimating things which fall within one order of magnitude [15][16]. Planning Poker as has been here defined was used in the empirical study reported in this article. It should be noted that no historical data was used when Planning Poker was employed in our study.

B. Analogy-Based Method (AbM)

The idea of using analogy as a basis to estimate effort in software projects is not new: in fact, Boehm [17] suggested the informal use of analogies as a possible technique thirty years ago. In 1988, Cowderoy and Jenkins [18] also worked with analogies, but they did not find a formal mechanism to select the analogies. According to Shepperd and Schofield [19], the principle is based on the depicting of projects in terms of their characteristics, such as the number of interfaces, the development methodology, or the size of the functional requirements. There is a base of finished projects which is used to search for those that best resemble the project to be estimated.

So, when estimating by analogy, there are *p* projects or cases, each of which has to be characterized in terms of a set of *n* characteristics. There is a historical database of projects that have already been finished. The new Project, the one to be estimated, is called "target". Such target is characterized in terms of the previously mentioned *n* dimensions. This means that the set of characteristics will be restricted to include only those whose values will be known at the time of performing the prediction. The next step consists of measuring similarities between the "target" and the other cases in the *n*-dimensional space [19].

Such similarities may be defined in different ways, but most of the researchers define the measuring of similarities the way Shepperd & Schofield [19] and Kadoda, Cartwright,

Chen & Shepperd [20] do: it is the Euclidean distance in an *n*-dimensional space, where *n* is the number of characteristics of the project. Each dimension is standardized so that all the dimensions may have the same weight. The known effort values of the case closest to the new project are then used as the basis for the prediction.

In our empirical study, we estimated effort by applying AbM in two distinct ways: our first approach was to take into account the characteristics of each user story in a general manner (AbM-1), and our second approach was to use only one characteristic of the n characteristics which could be used. In this case in particular, such characteristic was size (AbM-2s).

When using AbM-1 the participants compared the user stories of two projects: one considered "historical" and the other one "target". The Estimated Effort (EE) of the user story of the target project was, in fact, the Actual Effort (AE) of the "most similar" user story of the historical project.

When using AbM-2s, the project characteristic which was taken into account was the size of the project measured using COSMIC [21]. The EE of the user story of the target project was the AE of the closest historical user story –i.e., the user story whose size distance was the smallest ($|UserStorySize_t - UserStorySize_h|$**).** When multiple user stories were at the minimum distance, the participants calculated the mean of the AE of these user stories.

C. Historical Productivity

Jørgensen, Indahl, and Sjøberg [10] defined Productivity as the quotient of Actual Effort (AE) and Size, and the EE as the product of Size and Productivity. In this empirical study, COSMIC [21] was used as a measure of Size, and EE was calculated as the product of Size and Historical Productivity (HP). HP was the productivity of the project which was used as historical project, that is, the quotient of the AE and the Size of the historical project.

To measure size, COSMIC was selected because it is an international standard [22] that is widely recognized in the software industry, and also because there is a previous study that used it in an Agile context [23]. With the COSMIC software method, Functional User Requirements -possibly represented via user stories- can be mapped into unique functional processes. Each functional process consists of sub-processes that involve data movements. A data movement concerns a single data group, i.e., a unique set of data attributes that describe a single object of interest. There are four types of data movements:

- *Entry* moves a data group from a functional user into the software
- *Exit* moves a data group out of the software to a functional user
- *Read* moves a data group from persistent storage to the software
- *Write* moves a data group from the software to persistent storage.

In the COSMIC approach, the term "persistent storage" denotes data (including variables stored in central memory) whose value is preserved between two activations of a functional process. Moreover, the size of a software application is given by the sum of the sizes of its functional processes, and the size of the functional processes is given by the sum of Entries, Exits, Reads and Writes, where each term in the sum indicates the number of corresponding data movements, expressed in CFP. So, the concept of "weighting" a data movement does not exist in COSMIC; in other words, all data movements weigh the same.

IV. Definition and planning of our Empirical Study

Our empirical study is described in this section, considering its conception and how it was planned.

A. Definition

This empirical study was designed in order to establish when the accuracy of an expert estimation made in an agile development context, under the circumstances that will be described below, may be improved by using historical data. Such circumstances are: the project domain and the technological environment must be new to the estimator, and the team would have recently been created, so that the team velocity will be unknown.

The development steps of this empirical study may be summarized as follows:

The study was developed in the context of graduate education for IT practitioners from different educational and work backgrounds. The participants attended a workshop which had two objectives, one oriented to the subjects and another one oriented to the development of this empirical study. The workshop gave the participants the opportunity to: a. understand both how a historical database is built, and under which circumstances such database will give value to the estimation process, b. estimate using three methods and c. compare their results with other participants' results. Later on, the same workshop was conducted for undergraduate students.

The workshop participants were asked to re-estimate the first sprint of an application –the "target" application, i.e., P2- which had been previously developed by a group of undergraduate students who did not participate in the workshop. Both the development language and the application domain were unfamiliar to participants. Initially, participants had no idea of the developing team's velocity.

The re-estimations were performed by using four different estimation methods: ExE, based on the participants' intuition, and three other methods which use historical data. The historical data was obtained from an application which was similar to the target application, which had been developed by a third undergraduate group – a group that had neither developed the original application nor participated in our empirical study-. Such application, P1, will be called "historical application".

To guarantee the best results, we developed this empirical study following the recommendations of Juristo and Moreno [24] and Wohlin et al. [25]. To report it, we took into account Jedlitschka, Ciolkowoski and Pfahl's guidelines for reporting empirical research in software engineering [26].

As previously stated, the objective of this empirical study was to analyze when the accuracy of an estimation made by an expert, a role played by undergraduate students and practitioners in this study, may be improved by using historical data. This objective was achieved by comparing the errors the experts obtained by estimating with a method based on "pure" intuition (ExE) to those they obtained by estimating with three different methods: AbM-1, AbM-2s and HP.

Figure 1 summarizes this definition.

The hypotheses to be tested were:

H_0: The mean value of the MRE calculated with the ExE method is equal to the mean value of the MRE obtained when calculating with AbM-1, AbM-2s or HP.

H_1: The mean value of the MRE calculated with the ExE method is lower than the mean value of the MRE obtained when calculating with AbM-1, AbM-2s or HP.

B. Planning

The *experimental subjects* were IT graduate students and undergraduate advanced students of Informatics Engineering. In fact, all of the graduate students were practitioners. So, in this paper, when we say "participants" we mean both the graduate and undergraduate students, and by "practitioners" we refer only to the graduate students.

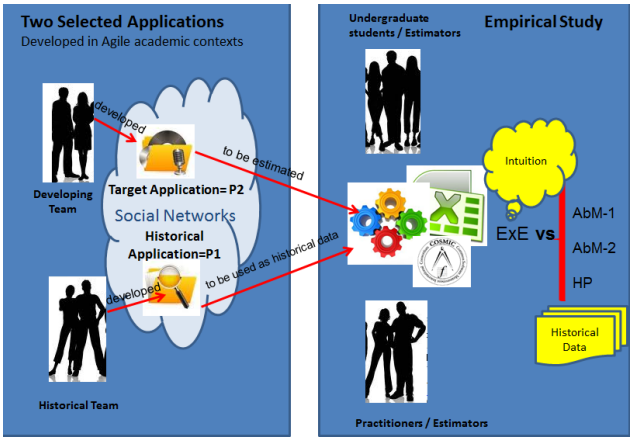


Fig. 1 Summary of the empirical study.

The participants were asked to give some information about themselves regarding the following aspects:

- If graduate or undergraduate student
- Professional experience (they had to state the number of years they had worked in software development)
- Experience with COSMIC
- Experience with user stories (they had to inform the number of user stories that they had written/read (fewer than 20, 20-100, more than 100)
- Experience with Ruby [27] language.
- Experience in Database development
- Experience in working in Agile development contexts.
- Level of prior knowledge about the productivity of the teams that developed the experimental objects (high, medium, low)
- Level of experience in the technologies used to develop the experimental objects (high, medium, low)
- Level of experience in the domain of the experimental objects (high, medium, low)

The *experimental objects were* two similar applications (P1 and P2), namely social networks, which had been developed by two groups of students before the empirical study was designed. For the sake of clarity we will say that P2 was developed by the "developing team" and P1 by the "historical team", as shown in Figure 1. It is important to note that these two groups did not participate in our empirical study; in fact, they were undergraduate students from a university different from the one where the undergraduate participants of the study were studying. P1 and P2 were developed to fulfill an assignment in a certain course. Both teams used Agile methodology to do so. They were instructed to register the hours worked per user story using the Scrumy tool [28]. Two professors supervised all of these tasks.

Application P1 is a system through which users may conduct surveys. The system classifies users into several categories, builds different groups and instantly surveys those users who fall within the right categories.

Application P2, which we have identified as the "target" project, is a network where different types of events may be published. For example, an event may be a party, a meeting or a football game. Events are the core elements in this application, not people. It works with event and friend suggestion algorithms and gives the option of buying a ticket for an event online.

The data corresponding to the experimental objects are displayed below. Table I shows the user stories of P1 and the Actual Effort (AE) of each user story measured in person hours. As some user stories were not functional processes, they were discarded. Table II shows the user stories and the AE of P2. This empirical study used the actual effort of P1 and P2. These are the user stories of only the first sprint, as it was the only sprint for which effort was estimated.

The aspects of the development process that were controlled to facilitate such comparison were:

- Similarity: Two similar applications that had been developed in Agile contexts were selected as experimental objects. They had been developed in an academic context by advanced undergraduate students, who had been requested to develop an application for an assignment in which a company environment was simulated.
- Experience in team velocity: Since in Agile contexts developers learn from previous estimations, and in this case the estimators were expected to have no previous experience, only the first sprint of the target application could be estimated in order to be compared to the actual effort estimation of P2, as it was only for the first sprint that the original P2 estimators did not have experience in team velocity.
- Language experience: Participants with experience in Ruby language, in Agile contexts, and / or COSMIC were equally distributed.

In order to obtain comparable results in this study, person-hours had to be used to unify the unit of measurement of effort, since the historical values had been previously measured in person-hours, instead of in story points or ideal hours, which are the measures usually used to make effort estimations with Planning Poker in Agile contexts [4].

The workshop was run following these steps:

1) The participants were given a set of materials that included: Brief Vision Documents [29] of P1 and P2, the professor's slides explaining the empirical study, and an Excel file where each sheet was a step of the empirical study.

2) Each one of the empirical study steps was explained to the participants. The participants were trained to perform each activity. Also, two examples of COSMIC measurement were included.

It is important to note that the participants worked with an Excel file that was designed to facilitate the understanding of the activities, and the sequence in which they had to do them. The following are the activities presented sequentially in each one of the sheets in the file:

a) *Perform the expert estimation*. Based on their intuition, they estimated the person-hours to be worked on the target application (P2). Based on the Vision Document of P2, the participants estimated the EE of each user story described in Table II.

b) Build the historical database. Each team created its own historical dataset by measuring the size of the user stories of the historical application (P1), using COSMIC, as shown in Table I.

TABLE I. USER STORIES OF THE HISTORICAL APPLICATION (P1)

User stories	Actual Effort [person-hours]
Create survey	18
Sign up	15
See user's profile	9
Answer survey	9
Log in/Log out	6
Comment on survey	12
Search for survey	9
Eliminate user	3
Edit personal data	6
Search for user	9
Generate and publish statistics	30
Follow user	30
Select user segment	18
Sort the content according to date	18
Upload pictures	21
UPR (User Popularity Ranking)	36

TABLE II. USER STORIES OF THE TARGET APPLICATION (P2)

User stories First Sprint	Actual Effort [person-hours]
Create, Modify and Eliminate User	8
Log in (Log out)	18
Create event	6
Search for event	3
Total	35

The Excel sheet automatically calculated the Historical Productivity (HP) of P1 as the quotient of AE_{P1} and $Size_{P1}$, where AE_{P1} is equal to the sum of the AE of each user story of P1, and $Size_{P1}$ is equal to the sum of the Size of each user story of P1. So, the HP_{P1} is calculated at application level, which will be used to automatically calculate the EE_{P2}.

The data movements of P1 were indentified for each user story, based on: the information included in the Vision Report, the name of the user story, and the explanation given by the leader of the workshop when asked for it. The measurement of the user stories, using COSMIC, was performed in a way similar to that of [23].

c) *Measure the size of the target application (P2), by using COSMIC to measure the size of the user stories.* These size values were automatically used to calculate $EE_{P2,}$ which was calculated as the product of $Size_{P2}$ and Historical Productivity (HP_{P1}), which had been obtained in the previous step.

d) *Estimate the effort for the target application (P2) using AbM in two different ways: AbM-1 and AbM-2s.* For AbM-1, the participants had to select for each one of the user stories in P2 the most similar user story from the set of user stories in P1 -though based on the stories' general

characteristics, not on their size or on any other specific characteristic- and then assign to the EE of each user story in P2 the AE of the similar user story in P1. For AbM-2s, the participants had to use the sizes which had been previously calculated for each one of the user stories in P2, as described in c) above, and the sizes which had been calculated for P1, described in b) above. They had to compare the size of each user story in P2 to the size of all the user stories in P1 in order to find the smallest distance between $|UserStorySize_{P2} - UserStorySize_{P1}|$. Once the smallest distance had been found, the AE of the user story in P1 which was the closest to P2 was assigned to the EE of such user story in P2. In those cases in which the participants found that the smallest distance was repeated, they assigned as EE of P2 the mean value of AE of the user stories in P1 which shared the same distance values.

e) *Individually compare and analyze the EE values obtained using ExE, AbM-1, AbM-2s and HP methods*. The Excel sheet automatically presents a table that displays the four EE values –those obtained by applying the four different estimation methods- for each user story in P2.

3) The participants estimated the effort of the target application following the steps listed above, and completed the worksheets.

4) The data was collected and the results were analyzed with the participants. A rich discussion about the comparison of the MRE obtained by applying the four estimation methods (ExE, HP, AbM-1 and AbM-2s) was conducted by the leader of the empirical study.

Figure 2 summarizes the steps of the empirical study.

C. Execution

Forty nine undergraduate students, who were distributed in fourteen groups of 3-4 students, participated in the two workshops. The median work experience of the students was three years. No one had experience using COSMIC, and they had little experience with user stories. All of them had attended the course "Database" and passed the exam and only 8 had experience in working in an Agile context, that is to say, a small proportion of them. The Level of experience of the development teams in the technologies to be used and in the domain of the experimental objects was low.

The characteristics of the participants are described in Table III.

We noticed that there were three aspects that affected the intuitive expert estimation: the work experience, the level of experience in the technologies used to develop the experimental objects, and the level of experience in the domain of the experimental objects. The undergraduate participants' work experience measured in years varied from 0 to 13, with a median of 3. This shows that the "experts" had little experience in estimations and also, that the level of experience in the technologies used and in the domain was low.

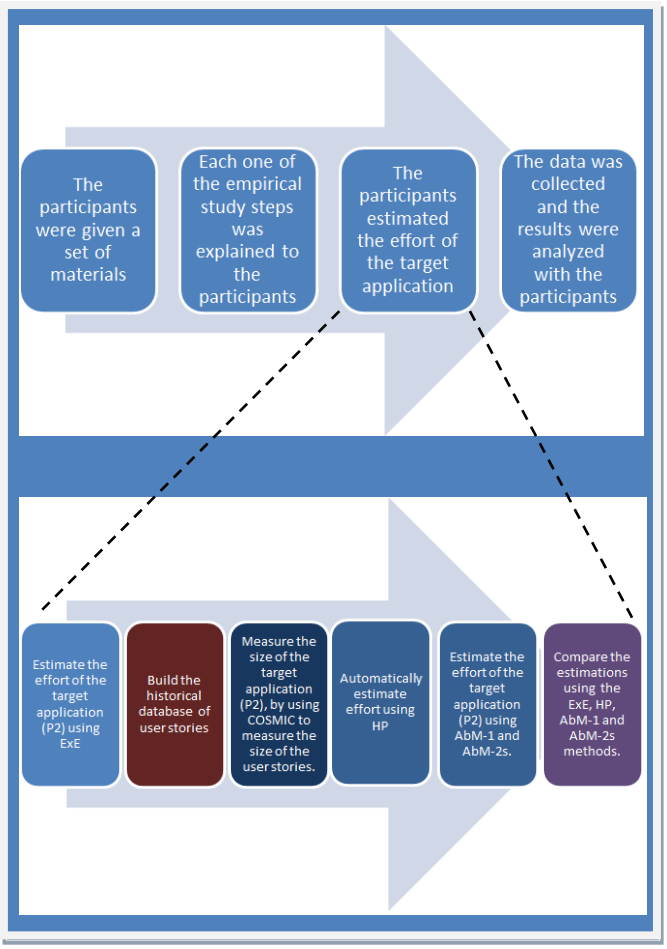


Fig. 2 Steps of the empirical study.

In one of the workshops, there were fourteen practitioners worked on their own and their median work experience was fourteen years. No one had experience in using COSMIC, and five of them had experience with user stories. Their median work experience with databases was ten years and only three of them had experience in working in an Agile context, which is a small proportion. The Level of experience in the technologies and in the domain of the experimental objects was medium-low, that is, not definitely low, but it could not be classified as fully medium.

When compared to the undergraduate participants, the most significant difference was their work experience: measured in years, it varied from 4 to 36, with a median of 14. Ten practitioners were project leaders or managers, three were senior developers and only one was a junior developer. This shows that these "experts" had experience in project management and, of course, in estimations.

V. RESULTS

Before answering the research question posed above, it is important to understand the circumstances under which

TABLE III. WORKSHOP PARTICIPANTS

Type	Number	Work Experience (Years)	Number of people familiar with COSMIC	Number of User Stories [<20, 20<US<100, >100]	Work experience with Database	Number of people familiar with Ruby Language	Number of people familiar with Agile context	Experience in the technologies	Experience in the domain
Undergraduate	49 (14 groups)	[0-13] Median: 3	No one	<20: 44 20<US<100: 3 >100: 2	All of them had attended the course "Database" and passed the exam	No one	Only 8	Low: 47 Average: 2 High: 0	Low: 43 Average: 4 High: 2
Practitioners	14	[4-36] Median: 14	No one	<20: 9 20<US<100: 3 >100: 2	Database experience measured in years [0-36] Median:10	Only one	Only 3	Low: 9 Average: 5 High: 0	Low: 11 Average: 3 High: 0

the use of historical data may improve expert estimation accuracy. To do so, this section will first describe the results obtained by the two types of participants -undergraduates and practitioners- and then analyze them. Afterwards, the statistical significance of such results will be dealt with, and later on, the research question will be answered. Finally, this analysis will be completed with the discussion of aspects omitted in the previous sections.

1) Description and analysis

Table IV shows the effort estimation values calculated for the target project, obtained by the two groups applying the four estimations methods: ExE, HP, AbM-1 and AbM-2s. Moreover, the AE of the target application (P2), which was developed by the undergraduate group was 35 person-hours.

Figure 3 shows the boxplots of the residuals and Figure 4 the boxplots of the MRE for the target project.

To obtain the MRE, the actual value registered for the first sprint of P2 by the group that actually developed the project was used as AE. Also, Table V shows the statistical functions of residuals and the MRE.

The boxsplots show the different results obtained by each group of participants. The undergraduate participants obtained better estimation results when applying the AbM-1 or AbM-2s, rather than the ExE or HP methods. Figure 4 shows the median values, but it must be noted that a more significant difference was observed when comparing the values obtained for the mean MRE in the undergraduate group: AbM-1: 0.70, AbM-2s: 0.71, ExE:1.51 and HP:1.75. On the other hand, the practitioners' group obtained the best results when applying ExE, instead of HP or AbM, as shown by the boxplots. Also, their mean values were ExE: 0.38, HP: 2.05, AbM-1: 0.87 and AbM-2s: 0.60.

The best result of the undergraduate group was obtained when using AbM: the MRE median of AbM-1 was 0.63 within the [0.37-1.83] range and AbM-2s was 0.62 within [0.13-1.14]. The lack of experience, in this case, was compensated for by the historical data. By using HP, the MRE dispersion was increased: the MRE values ranged from [0.99-2.72]. The MRE of the 14 groups had a median of 1.89 and a standard deviation of 0.54. Moreover, by using ExE we obtained a higher dispersion: MRE values ranged from [0.03-4.91], with a median of 0.90 and a standard deviation of 1.55.

Both the practitioners' level of experience in the technologies used to develop the experimental objects and their level of experience in the domain of the experimental objects were medium-low. These characteristics justify the results obtained when using ExE: the median of the MRE was 0.29 in a [0.14-0.83] range of values.

During the study, three of the practitioners assigned to the expert estimation the same value they had assigned to the AbM-1 estimation. This may have been a coincidence, or they may not have felt confident to perform an estimation based on their intuition.

Eleven out of fourteen practitioners obtained MRE less than 0.25 via ExE. The estimation by AbM-1 had a MRE median of 0.70 in a range result of [0.09-2.00] and that by AbM-2s obtained 0.51 in a range of [0.06-1.37], which are results similar to those obtained by the undergraduates. Moreover, the subtle differences between the MRE medians and the standard deviations of AbM-1 and AbM-2s may be justified by the fact that AbM-1 is based on subjective criteria, while AbM-2s is based on the concept of size. In fact, the practitioners had worked in very different contexts, which naturally affected their subjective comparisons.

TABLE IV. EE OF THE TARGET PROJECT

Participants	Number of estimations	Id Participants	ExE	HP	AbM-1	AbM-2s
Undergraduates	14 (made by groups of 3-4 undergraduate students)	1	161.00	110.00	57.00	-
		2	61.00	74.70	57.00	48.75
		3	34.00	76.30	60.00	75.00
		4	65.00	69.72	48.00	-
		5	207.00	84.12	48.00	-
		6	85.00	106.13	55.00	63.17
		7	173.00	90.21	66.00	52.03
		8	68.00	102.84	57.00	52.50
		9	79.00	101.15	57.00	56.79
		10	56.00	101.44	51.00	51.40
		11	51.00	72.00	57.00	72.00
		12	32.00	130.15	57.00	39.60
		13	105.00	108.93	99.00	73.83
		14	23	120.05	63	71.50
Practitioners	14	15	11.00	108.94	11.00	59.60
		16	30.00	173.22	24.00	45.00
		17	21.00	84.77	20.00	51.30
		18	30.00	122.15	60.00	60.73
		19	9.00	90.55	9.00	47.67
		20	64.00	85.96	39.00	57.00
		21	30.00	120.61	105.00	38.00
		22	29.00	111.87	86.00	82.80
		23	16.00	72.88	32.00	68.00
		24	30.00	105.05	95.00	52.75
		25	40.00	88.93	57.00	73.88
		26	40.00	97.07	94.00	52.50
		27	49.00	92.37	70.00	-
		28	57.00	140.94	57.00	37.00

TABLE V. STATISTICAL FUNCTIONS OF RESIDUALS AND MRE

Participants	Statistical Functions	Residuals				MRE			
		ExE	*HP*	*AbM-1*	*AbM-2s*	*ExE*	*HP*	*AbM-1*	*AbM-2s*
Undergraduate	Mean	-50.71	-61.27	-24.43	-24.69	1.51	1.75	0.70	0.71
	Median	-31.50	-66.30	-22.00	-21.79	0.90	1.89	0.63	0.62
	Standard deviation	56.40	18.80	12.43	12.03	1.55	0.54	0.36	0.34
Practitioners	Mean	2.43	-71.81	-19.21	-20.86	0.38	2.05	0.87	0.60
	Median	5.00	-66.06	-22.00	-17.75	0.29	1.89	0.70	0.51
	Standard deviation	16.25	26.30	32.65	13.35	0.26	0.75	0.61	0.38

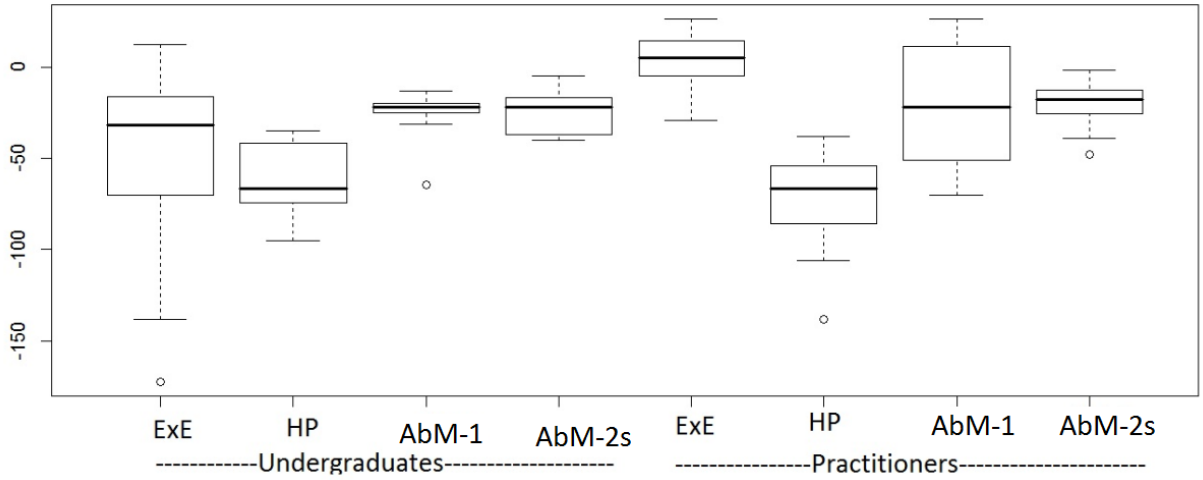


Fig. 3. Boxplots of the residuals of the target project.

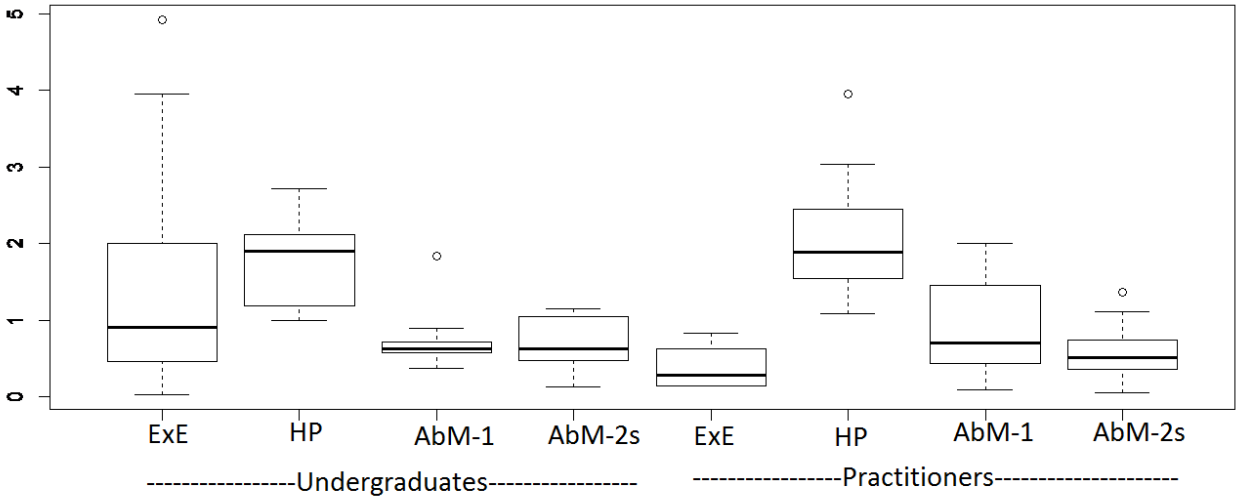


Fig. 4 Boxplots of the MRE of the target project.

By using HP, the MRE dispersion was increased: [1.08-3.85]. The MRE of the 14 practitioners had a median of 1.89 -similar to that of the undergraduate value-and a big standard deviation of 0.75, which may have been caused by the difference in productivity between P1 and P2.

2) Statistical significance

In order to test the hypotheses presented in Section III, the Wilcoxon rank test, at a significance level of 0.05, was used to analyze the statistical significance of our results. This non-parametric test was selected because the distributions of the variables were not normal. It was applied to test the accuracy of ExE versus that of HP, AbM-1 or AbM-2s, according to the results obtained by each group (practitioners and undergraduate participants). The MRE and the absolute residuals were used. Table VI shows the p-value of each subset, when using the MRE. The results obtained when using the absolute residuals are not shown because they presented no significant difference. When analyzing the MRE obtained by:

- the practitioners, when comparing ExE to HP, it was possible to reject H0 in favor of H1.
- the practitioners, when comparing ExE to AbM-1, once again, it was possible to reject H0 in favor of H1.
- the practitioners, when comparing the ExE method to AbM-2s, it was not possible to reject H0 in favor of H1.
- the undergraduates, when comparing the ExE method to HP, it was not possible to reject H0 in favor of H1.
- the undergraduates, when comparing the ExE method to AbM-1 and AbM-2s, it was not possible to reject H0 in favor of H1.

TABLE VI. STATISTICAL SIGNIFICANCE

Groups	ExE vs:	p-value
Undergraduate	HP	0.162
	AbM-1	0.948
	AbM-2s	0.793
Practitioners	HP	0.000
	AbM-1	0.022
	AbM-2s	0.083

The statistical significances obtained by the practitioners using AbM-1 and AbM-2s are similar, but it was not possible to reject H0 in favor of AbM-2s, although we did reject it for AbM-1. The main reason is the differences in the distributions between AbM-2s and AbM-1, as shown in Figure 4. Besides, the calculation may have been affected by the lower number of available instances.

It should be noticed that the EE values reported by the three practitioners who presented the same values when using ExE and AbM-1 were also included in the table. However, later on, when the Wilcoxon rank test was run, we only considered the values reported by the other eleven practitioners, and the results did not vary.

Now we can answer the research question: W*hen may the accuracy of an expert estimation made in a context of Agile software development be improved by using historical data?*

These results show that the expert estimation was not improved by the use of historical data when the expert had some work experience, and his level of experience in the technologies used to develop the application together with his level of experience in its domain were medium-low.

However, we have found out that historical data may improve expert estimation when the estimator's work experience, his level of experience in the technologies used to develop the application, and his level of experience in the domain of the application to be developed is low.

1) Discussion

There are some aspects that have not been mentioned yet, but we believe they are worth being discussed at this point. One of them is the little experience in Agile development contexts that the two groups had. We think that this fact did not affect the results obtained because, although the work experience of the undergraduate group was limited, so was their experience in Agile contexts. On the other hand, the fact that practitioners were experienced in project management and estimations compensated for their little experience in Agile contexts. Furthermore, as the empirical study was designed to only use the first sprint of a software product development, no estimations were made for the rest of the sprints -which would be usually done when using an Agile method- so their little experience in Agile contexts had no impact on our study.

Another interesting aspect is that most of the effort calculations proved to be underestimated, which may be seen in Figure 3. This could be explained by the fact that almost all the participants did not have previous experience with the Ruby language.

One question that may arise is: how would the participants be able to make meaningfully expert estimations if they did not have any knowledge about the developers? This condition was part of the scenario that we were simulating; as stated in the introduction of this paper, the team velocity would be unknown.

Figure 4 shows that the medians obtained by the two groups when estimating with HP were similar, but their standard deviations were not: the standard deviation of the MRE for the undergraduate group was 0.54 and 0.75 for the practitioners. The estimation was affected by the subjectivity of the measurement which may be explained by the differences between means and median, for the two groups: a. undergraduate: mean: 62, median:62 and b. practitioners: mean:56, median:53. In Table VII and VIII the measurements made by each group of participants are reported. It is important to note that the standard deviations are quite similar: 12.25 and 11.97.

TABLE VII. MEASUREMENTS MADE BY UNDERGRADUATES USING COSMIC

Id User Story	2	3	6	7	8	9	10	11	12	13	14
1	10	4	6	5	4	4	5	6	3	7	4
2	7	5	4	5	3	4	4	5	3	4	5
3	2	3	2	3	2	3	3	3	2	3	2
4	11	3	4	6	4	4	4	5	3	3	3
5	3	4	3	5	4	5	5	5	3	3	4
6	5	3	4	4	2	3	4	4	2	3	3
7	3	4	3	4	3	4	3	5	3	4	4
8	8	5	4	5	2	5	3	4	3	3	4
9	4	5	4	5	4	6	4	6	4	3	4
10	3	4	3	4	3	3	3	3	3	4	4
11	7	6	5	5	3	6	4	11	3	8	3
12	4	4	4	5	2	4	3	5	2	3	5
13	4	4	4	4	3	4	2	3	2	3	3
14	3	1	3	3	3	2	3	5	3	3	3
15	4	5	4	4	2	4	2	5	3	4	3
16	2	2	4	2	2	3	2	8	2	6	2

TABLE VIII. MEASUREMENTS MADE BY PRACTITIONERS USING COSMIC

Id User Story	14	15	16	17	18	19	20	21	22	23	24	25	26	28
1	4	3	8	3	3	2	8	4	4	3	4	4	3	2
2	5	3	1	3	3	2	4	4	3	2	4	4	4	3
3	2	3	2	3	3	2	4	3	3	3	3	3	3	3
4	3	2	2	2	4	2	5	5	5	3	6	3	5	4
5	4	4	4	5	3	5	6	5	5	2	5	4	3	5
6	3	2	2	2	3	2	7	5	4	2	5	2	4	4
7	4	3	3	3	3	3	5	3	4	2	3	3	3	2
8	4	4	4	2	3	2	7	5	4	2	5	3	5	4
9	4	3	2	4	6	3	4	5	5	3	5	4	5	4
10	4	3	2	3	3	3	3	4	4	3	4	3	4	3
11	3	3	2	3	3	3	9	3	6	3	3	8	3	3
12	5	4	2	2	3	3	4	4	4	2	4	4	4	4
13	3	3	3	3	3	3	5	4	5	3	4	3	3	3
14	3	3	3	3	3	3	3	3	5	3	3	3	3	3
15	3	2	3	3	3	3	6	5	4	2	4	2	5	4
16	2	3	3	3	4	3	4	2	4	3	2	3	2	2

Some of the measurements (1, 4, 5, 27) are not reported because they are not available.

Figure 4 shows that the MRE medians obtained when the two groups used the AbM-1 method were similar but their standard deviations of MRE were quite different. The practitioners' standard deviation was bigger than the undergraduates' standard deviation. This may be a consequence of the variety of persons that made up the practitioners' group: when they had to select the "most similar" user story, they applied their own criteria, based on their different work experiences, which were definitely subjective.

On the other hand, the undergraduates and practitioners got similar distributions with AbM-2s. The reason may be that the two groups used an objective measure of size: COSMIC.

It was not surprising that the results obtained with HP were clearly worse than those obtained with AbM. This was expected, since HP is a method that estimates at application level, while AbM estimates at user story level.

The estimation results obtained with the AbM and HP methods would have been better if the historical data had been obtained from a similar project –one developed using Ruby on Rails-, but unfortunately, there was none available. Besides, the fact that the user stories that were not functional processes were discarded may have also influenced the results. In addition, another interesting factor that may have been considered is team size.

In our study, the empirical objects were two similar applications, but what would have happened if they had not been similar? Obviously, the results of the undergraduate group would have been affected, as their best results were obtained using AbM. The reason is that such method is based on analogy, so if the degree of similarity between the application from where the historical data was to be obtained and that of the target application had been low, the accuracy of the estimation would have been poor too.

Moreover, although we only used the estimates of the first sprint of the target application this time, we believe the estimates of the following sprints could be used in future replications to evaluate if (and to what extent) expert estimations improve while participants gain knowledge of the projects (while AbM and HP are expected to yield constant accuracy throughout the sprints).

Finally, we may wonder about the participants' characteristics included in Table III and the reason why other characteristics were not included. To begin with, database experience is related to work experience, so it was necessary to check it because the COSMIC measurement would have been affected if experience in database had been small. In fact, the experience in using COSMIC was defined as a controlled variable. Moreover, the number of user stories the participants had written/read was included because it is related to their work experience in Agile contexts: in fact, there was a correlation between the number of user stories read/written and their experience in Agile contexts, which proved the consistency of the information. In addition, the level of experience with Rugby language and the level of experience in the technologies to be used had to be tested in order to verify if the participants fit our empirical study. Furthermore, the impact of the level of experience in the application domain was previously analyzed by [31]. We think that these characteristics have made the main differences between the two groups clear.

VI. THREATS TO VALIDITY

The difference in the background of the experimental subjects is the major weakness of this empirical study. However, this drawback may be transformed into a strength if we consider that in this empirical study the experience of the expert is stressed, showing that the accuracy of an expert estimation depends on the estimator's expertise, which is measured by his work experience, his level of experience in the technologies used to develop the experimental objects and his level of experience in the domain of the experimental objects.

The productivity rate of academic developments is usually quite different from the one of professional settings. This fact has obviously affected the results obtained, but as it is reflected in the error values, it does not invalidate the empirical study.

Another threat is that the expert estimations were made in two different manners: either alone or in groups. The practitioners worked alone and the undergraduate students formed groups of three or four persons and used Planning Poker to obtain the expert values. In spite of this difference, we think that combining expert methods, that is, using Planning Poker or not, did not introduce bias in this study, in accordance with what was reported in [30].

Unfortunately, only a brief explanation about COSMIC was given to the undergraduate students since there was not enough time to give an extensive explanation (the whole workshop was three hours long). Thus, the little available time was devoted to those COSMIC characteristics that were necessary for them to know in order to make a correct measurement. However, this did not seem to be a serious problem, as the concept of data movement was quite intuitive for all the participants and the medians of the errors shown in both Figure 3 and 4 for the HP method are similar.

Also, the use of examples and previous training in Function Points made it easier for the participants to understand how to use this measuring method. On the other hand, the practitioners had been previously trained in COSMIC, so they presented no difficulty. Besides, if anybody had any doubts, the person who led the empirical study gave them further explanations.

The order in which the estimations were performed may have introduced bias in the result, so it would have been more convenient if the participants had not performed the estimations in the same order, except for ExE, which must always be performed in the first place.

When building a historical database, the selection of an application similar to the one to be estimated is clearly an advantage in order to obtain a better estimation. In this empirical study, we used as historical application one that had not been developed in the same language the application to be estimated had been. Obviously, this circumstance may have enlarged the estimation error of the method that used historical data. At the same time, as the context defined for this empirical study was one in which the project domains and the technological environments were new to the team, we interpreted that the application used as historical was well selected.

The accuracy of AE registered by the students that developed P2 was controlled by two professors. In fact, the students registered in a web application the user stories and the tasks done, the EE and the AE of each user story and the EE and AE of each task assigned to each user story. This detailed registration facilitated the control for accuracy.

The experimental subjects were identified either as undergraduates or practitioners. However, it may be argued that more categories would have been necessary, as some of the practitioners had more experience in the domain or in the technologies than some others. Consequently, to obtain more evidence of the benefit of using historical data, it is necessary to have a bigger number of estimators, which would allow us to identify different levels of expertise, for example, three expertise levels for practitioners and three for undergraduates.

To conclude, as the experimental objects used in the empirical study came from only one particular environment and the experts' experience did not cover the big spectrum of expertise that exists, general conclusions cannot be drawn because there may be different estimation problems in different environments and experts' performances.

VII. Conclusion and Future work

This paper specifically focuses on an agile context in which the project domain and the technological environments are new to the estimators, the teams have recently been created, and the team velocity is unknown. In our study the estimations performed by two different groups –undergraduate participants and practitioners- which first used intuitive expert estimations (ExE) and then three different estimation methods which use historical data (HP, AbM-1 and AbM-2s) were compared in order to find out whether there is any advantage in using historical data under these circumstances.

We may conclude that historical data seems to be valuable when the work experience, the level of experience in the technologies to be used to develop an application, and the level of experience in the domain of the application to be developed are low.

Consequently, for estimators who have the restrictions described above, and who have no option but to work with them, we may suggest the following:

- Use intuitive expert estimations when your work experience, your level of experience in the technologies to be used to develop the application, and your level of experience in the domain of the application to be developed are not low.
- Use historical data when your work experience, your level of experience in the technologies to be used to develop the application, and your level of experience in the domain of the application to be developed are low.

As historical data is not frequently available [32], we expect the results of this empirical study may motivate novice developers to give importance to collecting such data in their daily work.

In order to generalize this conclusion, a replication of this empirical study is recommended, especially if different software life cycle models [33], application domains, expert profiles, and levels of performance are included. Also, different estimation methods, such us linear regression may be used. Finally, in order to enrich this empirical study, it would also be convenient to compare an estimation performed by an expert who has deep knowledge of this domain, and also knows the team velocity, to the estimations obtained by the participants of our study.

AKNOWLEDGMENTS

Our thanks to the Research Fund of Austral University, which made this study possible, and to Luigi Lavazza for his opportune comments.

REFERENCES

[1] G. Robiolo, S. Santos, and B. Rossi, "Expert estimation and historical data: an empirical study," in Proceedings of The Eighth International Conference on Software Engineering Advances, ICSEA 2013, October 2013, pp. 336-345.

[2] M. Jorgensen, "A review of studies on Expert estimation of software development effort," Journal on System and Software, vol. 70, no. 1-2, 2004, pp. 37-60.

[3] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," IEEE Transactions on Software Engineering, vol. 33, no. 1, January 2007, pp. 3-53.

[4] M. Cohn, Agile Estimating and Planning. Addison-Wesley, 2005.

[5]O. Ktata and G. Lévesque, "Designing and implementing a measurement program for Scrum teams: what do agile developers really need and want?," in Proceedings of the Third C* Conference on Computer Science and Software Engineering (C3S2E '10), ACM, 2010, pp. 101-107.

[6] V. Mahnič and T. Hovelja, "On using planning poker for estimating user stories," J. Syst. Softw., vol. 85, no. 9, September 2012, pp. 2086-2095.

[7] S. Halstead, R. Ortiz, M. Córdova, and M. Seguí, "The impact of lack in domain or technology experience on the accuracy of Expert effort estimates in software projects," in Proceedings of the 13th international conference on Product-Focused Software Process Improvement (PROFES'12), Springer-Verlag, 2012, pp. 248-259.

[8] M. Jorgensen, and T. Halkjelsvik, "The effects of request formats on judgment-based effort estimation," Journal of Systems and Software, vol. 83, no.1, 2010, pp. 29-36.

[9] M. Jorgensen and M. Gruschke, "The Impact of lessons-learned sessions on effort estimation and uncertainty assessments," Software

Engineering, IEEE Transactions on, vol. 35, no. 3, 2009, pp. 368 - 383.

[10] M. Jørgensen, U. Indahl, and D. Sjøberg, "Software effort estimation by analogy and regression toward the mean," Journal of Systems and Software, vol. 68, no. 3, 2003, pp. 253-262.

[11] T.J.Bang, "An Agile approach to requirement specification," Agile Processes in Software Engineering and Extreme Programming, Springer Berlin Heidelberg, 2007, pp. 193-197.

[12] J. Choudhari and U. Suman, "Phase wise effort estimation for software maintenance: an extended SMEEM model," in Proceedings of the CUBE International Information Technology Conference, ACM, 2012, pp. 397-402,

[13] N.C. Haugen, "An empirical study of using Planning Poker for user story estimation," Proceedings of AGILE 2006 Conference, Computer Society, IEEE, 2006, 9 pp. – 34.

[14] J. Grenning, "Planning Poker or how to avoid analysis paralysis while release planning," 2002, doi: http://sewiki.iai.uni-bonn.de/_media/teaching/labs/xp/2005a/doc.planningpoker-v1.pdf: May, 2014.

[15] E. Miranda, "Improving Subjective estimates using paired comparisons," IEEE Software, vol. 18, no.1, 2001, pp. 87–91.

[16] T. Saaty, Multicriteria decision making: the Analytic Hierarchy Process. RWS Publications, 1996.

[17] B. Boehm, Software Engineering Economics. Prentice Hall, 1981.

[18] A.J.C. Cowderoy and J.O. Jenkins, "Cost estimation by analogy as a good management practice," in Proc. Software Engineering 88, Second IEE/BCS Conference, 1988, pp. 80-84,

[19] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," IEEE Trans. on Software Eng., vol. 23, no. 11, 1997, pp. 736-743.

[20] G. Kadoda, M. Cartwright, L. Chen, and M. Shepperd, "Experiences using Case-Based Reasoning to predict software project effort," Proceedings of the EASE conference keele, UK., 2000.

[21] COSMIC – Common Software Measurement International Consortium, The COSMIC Functional Size Measurement Method - version 3.0. Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2003), 2007

[22] ISO, IEC19761:2011, Software Engineering -- COSMICFFP– A Functional Size Measurement Method, ISO and IEC, 2011.

[23] J. Desharnais, L. Buglione, and B. Kocatürk, "Using the COSMIC method to estimate Agile user stories," in Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement, ACM, 2011, pp. 68-73.

[24] N. Juristo and A.M. Moreno, Basics of Software Engineering Experimentation. Kluwer Academic Publishers, 2001.

[25] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslen, Experimentation in Software Engineering: an Introduction. Kluwer Academic Publisher, 2000.

[26] A. Jedlitschka, M. Ciolkowoski, and D. Pfahl, "Reporting experiments in Software Engineering," in Guide to Advanced Empirical Software Engineering, Section II, 2008, pp. 201-228.

[27] Ruby on Rails, doi: http://rubyonrails.org/: May, 2014.

[28] Scrumy, doi: http://www.scrumy.com: May, 2014.

[29] K. Bittener and I. Spence, Use case Modeling. Addison Wesley, 2003.

[30] K. Molokken-Ostvold, N.C. Haugen, and H.C. Benestad, "Using planning poker for combining Expert estimates in software projects," Journal of Systems and Software, vol.81, no.12, 2008, pp. 2106-2117.

[31] M. Jorgensen, "Selection of strategies in judgment-based effort estimation," Journal of Systems and Software, vol. 83, no. 6, 2010, pp.1039-1050.

[32] C. Mair , M. Shepperd, and M. Jørgensen, "An analysis of data sets used to train and validate cost prediction systems," ACM SIGSOFT Software Engineering Notes, ACM, vol. 30, no. 4, 2005, pp.1-6.

[33] A. M Davis, E. H. Bersoff and E. R. Comer, "A strategy for comparing alternative software development life cycle models", Software Engineering, IEEE Transactions on, vol. 14, no.10, 1988, pp. 1453 – 1461.

An Ontology-Driven Personalization Approach for Data Warehouse Exploitation

Lama El Sarraj
LSIS UMR 7296
Aix-Marseille University
Marseille, France
lama.elsarraj@lsis.org

Bernard Espinasse
LSIS UMR 7296
Aix-Marseille University
Marseille, France
bernard.espinasse@lsis.org

Thérèse Libourel
Espace-Dev UMR 228
Montpellier 2 University
Montpellier, France
therese.libourel@univ-montp2.fr

***Abstract*— Data Warehouses (DW) resources are shared by users' from different backgrounds (e.g., domain, culture, education, profession). Those resources (e.g., OLAP queries, Excel files) are interpreted differently from a user to another. Unfortunately, misinterpreting data could induce serious problems and conflicts. To guarantee relevant interpretation of resources, additional semantic description of resources concepts is necessary. In this context, we present an Ontology-driven Personalization System (OPS) based on three connected ontologies: domain ontology, DW ontology and resources ontology. OPS return a set of personalized resources search based on users' domain and his recurring interests. In addition, resources are enhanced with a semantic description provided by the ontologies. This paper focuses on the methodology used to develop connected ontologies used by OPS.**

Keywords-data warehouse, ontology, personalization, decision support systems, decision making, healthcare institution management.

I. INTRODUCTION

Decision Support Systems (DSS) enables users to analyze and synthesize data according to different perspectives. Big companies need efficient DSS and seek to expand the number of their users. In fact, companies need to have flexible decision tools that include users' requirements and resources (e.g., Excel files, graphs, tables). Resources are shared by users' from different backgrounds (e.g., domain, culture, education, profession). Thus, resources interpretation depends on user backgrounds. We proposed in El Sarraj *et al.* [1] to use an ontology-driven personalization approach to facilitate the exploitation of Data Warehouse (DW) resources.

Generally a DSS uses a collection of Business Intelligence (BI) tools and applications to analyze, query and visualize a big volume of data from heterogeneous sources and domains stored in a DW. DW is the core of most DSS, it is "a subject oriented, nonvolatile, integrated, time variant collection of data in support of management's decisions" [2]. DW uses a multidimensional model that represents facts and their measurements, related to different dimensions, which are the axes of analysis. To facilitate the task of DW analysis and treatment, a subset of the DW is created, called Data Mart (DM). A DM is oriented to a specific business need or a particular user requirement. Most of the times, data mart are organized in a multidimensional structure [3]. Data are represented as a point in a multidimensional space, visualized like a data cube [4]. They give users the possibility to synthetize and analyze data from three (or more) dimensional arrays of values and various granularity levels. Based on this multidimensional model On Line Analytical Processing (OLAP) cubes enable the manipulation of data provided by the DW. In this paper, only the multidimensional table resource is considered.

In the DW field, taking user requirements into account is crucial for the success or the failure of the DW [5], especially when users belong to different domains. The exploitation level of DW, as well as the preliminary conception level, is mainly based and adapted to user requirements [6]. Most research works devoted on DW focuses on the design approach [7], [8], [9]. Even if these approaches are successful at the conceptual level knowledge about the DW resources is still needed. It is important that users understand the semantic of the information they analyze and have a visibility about other resources that could help them make efficient analysis.

Ontologies have already proved their utility to resolve semantic problems in DW domain. Ontologies are widely used in the DSS domain. First, they were used for DW design to facilitate the integration of data from heterogeneous sources. Indeed, DW are considered as data integration systems [10]. Then, researchers in this domain have widely used ontologies in different phases of the DSS, at the conceptual level [11], [12], at the Extract-Transform-Load (ETL) level [13], OLAP cube model [14] and OLAP queries [15].

The goal of this work is to develop an ontology-driven system for DW personalization to support users of various profiles to efficiently exploit a DW using existing DW resources. This paper focuses on the knowledge base component of such a personalization system. This knowledge base is composed of three ontologies: the first one is the domain ontology, the second one presents the schema of an existing DW, and the last one describes existing DW resources of a related DW.

This research concerns an existing DW used in the context of the "Program of Medicalization of Information Systems (PMSI)" to analyze healthcare institutions

activity. The PMSI is part of the reform of the French health system. The PMSI is a device that enables quantifying and standardizing the data about the healthcare institutions activity. PMSI data are used to finance healthcare institutions according to their activity. This research has been financed by the public hospitals of Marseilles - Assistance Publique des Hôpitaux de Marseille (APHM).

The paper is organized as follows. Section II presents related works about DW personalization and introduces some elements about ontologies. Section III presents the problematic. First, it introduces the context of this research with a case study from healthcare domain. Then, it presents the aim of this research. Section IV presents the general architecture of the "ontology-driven personalization system" and the uses-cases supported by this system. Section V presents the methodology used to develop the knowledge base of the personalization system. Also, it presents in details the knowledge base component, the type of knowledge concerned, the models in UML and OWL of the three ontologies: Domain Ontology (O_D), Data Warehouse ontology (O_{DW}) and existing resources ontology (O_R). Section VI presents the mapping between knowledge base ontologies. Finally, we conclude and present some perspectives to this work.

II. RELATED WORK

This section presents different researches related to DW personalization approaches, which are mainly based on users' profiles and recommendation techniques. This section also introduces some elements related to ontologies and their use in software development.

A. DW personalization based on user profiles

Researches works, based on user profiles, are usually associated to the "personalization" of DWs. After introducing and defining the concept of personalization in the context of DW, we present various existing approaches related to DW personalization. Then we compare and evaluate their relevance to our problem with the use of a DW.

Personalization is a customized and individualized description of a user or a group of users. Personalization system relies on users' need, preferences and characteristics [16], and usually on a defined users profiles [17]. Although no consensus exists for the definition of a user profile, but a profile generally includes a set of features that is used to configure or adapt the system to the user. Thereby, the system provides personalized and efficient results [18] adapted to a user profile.

The authors in [19] developed a state of the art about user modeling based on system requirements. Other researches configure or adapt the personalization system to users' preferences defined in their profiles [20], [6], [21]. These preferences may be related to their *contexts* defining application frameworks, as proposed in some researches in the DW domain [21], [22].

Bentayeb *et al*. [23], characterize the personalization of a DW based on user's profile from two perspectives, the definition of users profiles, which can be explicit or implicit, and the exploitation of these profiles to personalize the DW treatments:

- *Explicit implication* of the user at the profile definition level mainly needs to set parameters related to the recommendation process.
- *Implicit implication* of the user creates automatically a group of users profile based on a learning method and leads to an automatic transformation of the system.

The explicit definition is related to the configuration (customization, user modeling) and the implicit definition is related to adaptation (user profiling). In both cases, the profile may be operated by *recommendation* or by *transformation*, with automatic processes.

Jerbi *et al.* [24] distinguish three main objectives from DW personalization researches:

- *Customizing data sources schema* [22], [23], adapting the data structures to a specific needs of users.
- *Customizing queries visualization* [20], or representation [6], [21], [25].
- *Recommendation of OLAP queries* [26], [27] to assist in the exploration of the DW.

The first two objectives seem to affect data-centric personalization, in the first case by customizing the schema and in the second case by representing customized queries results. The third objective concerns the recommendation of a new method to treat data, queries.

B. DW personalization by recommendation or transformation

The personalization of the DW by recommendation is treated by various works such as [23], [26], [27], [28], [29], [30], [31], [32]. In these works we can distinguish two categories of recommendation methods: methods based on the *content* and methods based on *collaborative filtering*. The methods based on contents recommend similar objects. This recommendation is based on previous user actions while the methods based on collaborative filtering recommends items based on the interest and similar user.

The personalization of the DW by transformation, is mentioned by the authors in [20] that treats personalized visualization of OLAP queries. The authors in [33] propose a solution to evolve the DW schema according to user requirements. This method is based on "if-then" rules. The research work in [34] propose a solution to expand the DW architecture with event/condition/action rules. Finally, the authors in [21] propose customized

OLAP tables, based on users preferences and on analysis context.

To the best of our knowledge, no research uses ontologies to facilitate the exploitation of a DW. However, in our approach, we propose an ontology-driven personalization approach to facilitate the DW exploitation. The aim of our research is presented in details in Section III.

C. Ontologies

Ontologies have been used in the domain of knowledge engineering to facilitate requirements expression and detect incoherencies and semantic ambiguities between users [35]. Description Logic (DL) is a formalism used to build ontologies [36]. In this section, we define and propose a formalization based on DL for ontologies.

The first goal in the expected ontology is to provide resources to achieve automatic process, whether for machines interaction and interoperation with each other or with humans. Ontologies are used in several domains to resolve syntactic and semantic heterogeneity problems. In the software engineering field, ontology had been used first in the field of artificial intelligence systems and knowledge base systems, and then adapted to the problems of information retrieval. The use of ontologies in software engineering adds a wealth of knowledge to the systems.

Ontologies design requires the establishment of processes to extract the knowledge connected to a domain and make it suitable for both information systems and humans. In this context, several definitions of ontologies have been proposed in the field of software engineering. Gruber [37] defines ontology as a specification of a conceptualization "*[...] A conceptualization is an abstract, simplified view of the world that we want to represent*". This definition was extended by [38], which focuses on the formal characteristic of an ontology.

In our work, we consider the definition proposed by Jean *et al.* [39] a definition that characterizes an ontology as a referencing formal representation and consensus of all shared concepts. In this definition, the most important terms are:

- *Formal*: the ontology is interpretable by machines.
- *Explicit*: all concepts and properties of ontology are explicitly specified independently of any particular point of view or implicit context.
- *Referenceable*: any concepts described in the ontology can be referenced in a unique way from any context, in order to clarify the semantics of the referenced item.
- *Consensual*: the ontology is recognized and accepted by all the members of a community.

D. Formalization of the ontology

There's different existing languages to define ontologies. Ontology Web Language (OWL) is the standard language for representing ontologies [40], [41]. W3C consortium recommends OWL to define ontologies. The OMG [41] define the OWL meta-model.

DL language present the formalism underlying OWL language [36]. In DL, structured knowledge is described using concepts and roles. Concepts represent sets of individuals, and roles represent binary relationships between individuals.

A knowledge base described with DL is composed of two components: the Terminological Box (TBox), and the Assertion Box (ABox). The TBox specifies the intentional knowledge of the modeled domain.

In general, terminological axioms have the form of inclusions ($C \sqsubseteq D$) or equivalence ($C \equiv D$) such as (C, D denote concepts or roles).

Based on this definition, the ontology is formalized as 5-uplet [42] as follows:

O: <C, P, ClassPropt, ClassAssoc, Formal> such as:

- *C* represents the classes of the ontological model.
- *P* represents the properties of the ontological model, and P is partitioned into:
  - P_{value}: represents the characteristics properties.
  - P_{fct}: represents domain dependent properties.
- *ClassPropt*: $C \rightarrow 2P$ relates each class to its property.
- *ClassAssoc*: $C \rightarrow$ (Opr, Expr (C)) is an expression that associate to each class an operator (inclusion or exclusion) and an expression to other classes.
- *Formal*: is the formalism followed by the ontology model (e.g., RDF, OWL).

To facilitate the creation and the visualization of ontologies there are OWL ontology editors, such as Protégé [43] that manipulates ontologies (e.g., edit, load, define taxonomies). Protégé provides a detailed view for each concept in ontology. There are also visualization tools of ontologies, the most common ones are IsaViz [42], OWLViz [10], Growl [44], Welkin [39].

UML is a standard used to model information systems and software engineering. UML is a semi-formal formalism. UML is a graphical language for visualizing, specifying and building tool components. UML provides different diagrams (e.g., class diagrams). However, UML is not suitable to represent complex reasoning and inferences [46]. One of the major advantages of UML is that it is widely used in the academic environment and even by non-professionals. UML notations facilitate the knowledge visualization, especially of ontologies. Most informatics designers use UML to describe their diagram.

Several studies propose to model ontologies with UML [45], [46], [47], [48]. There are many commonalities between the formal languages of ontologies and UML. A comparison UML/OWL is studied [46], but the only drawback is the lack of semantics in UML. For those reasons, we can consider

UML as an adequate formal model for the representation of ontologies.

The process to create an ontology can be accomplished by: (i) modeling the ontology with UML to have a consensus between different users' experts, (ii) transforming UML model into an OWL model, to reason on the ontology.

E. Conclusion

Even if the personalization of DW is a recent field of research, various studies propose methods to treat this problem. In their study, the authors in [24] compare different works of DW personalization domain, they take in consideration three main aspects: (i) personalization objectives, customized schema or queries (the result or the visualization), (ii) user model type, that has been selected to define the user (rules, scores, preferences, annotations) and his contextualization, (iii) the algorithms implemented for DW personalization. These approaches do not seem totally adapted to our problematic. Indeed, the specificities of data exploited by a big number of users' from different backgrounds, require additional semantic to describe resources provided by DWs. We present in the next section the context and the aim of this work.

III. PROBLEMATIC

This section presents the context of our research introduced by a case study presenting a DW schema. Then we present the aim of our research.

A. Context

The application context of our research concerns the healthcare management applied specifically in the Program of Medicalization of Information Systems (PMSI) supported by the French government. In fact, PMSI is a French adoption of the concept created by the Professor R. Fetter (Yale university, United States of America) to finance hospitals. PMSI specifies the cost of sojourn based on the Diagnosis Related Groups (DRG) that classifies the hospitalization in homogeneous and coherent medico-economic groups. Today, this classification technique is used in France to finance healthcare institutions according to their activity.

To analyze PMSI data, specific DSS have been developed. DSS is mainly dependent on DW, and is used by different profiles of users. We identified two types of users profiles, the first type is related to a medical domain (e.g., doctors, pharmacists, biologists), while the second one does not (e.g., financial affaire managers, computer scientists, human resources).

In this context, in order to illustrate our problematic we consider the DW star schema presented in Fig. 1. This DW contains data concerning "PMSI activity". This DW schema is composed of a fact table, dimensions, and measures:

- Fact table = {F_Activity}
- Dimensions = {D_Time, D_Hospital_Structure, D_International_Classification_Of_Diseases, D_Exit_Mode, D_Diagnosis_Related_Groups D_Age}
- Measures = {Number of patient, Number of beds}

Note that a *pole of activity dimension* "Pole" is a set of medical services units. A *hospital structure dimension* "D_Hospital_Structure" is a set of "Poles".

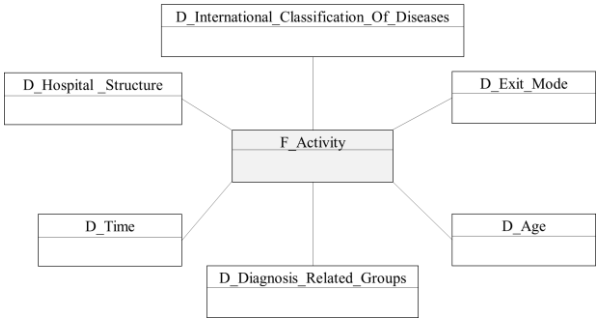


Figure 1: PMSI activity, DW Schema.

In this paper, we take the example of a Multidimensional Table (MT) (MT is defined in Section V.B) is denoted MT = (M, D), where M is the set of measure and D is the set of dimensions. We take an example of a multidimensional pivot table, presented in Fig. 2. For confidentiality issues this table is presented with fictive data:

- D1 = D_Hospital_Structure (dimension level "Pôle").
- D2 = D_Diagnosis_Related_Groups (attributes: DRG, TYPE DRG TITLE).
- M1 = number of patients (calculated measures: total of M1 per "Diagnosis Related Groups", total of M1 per pole, total of M1 for all Diagnosis Related Groups (i.e., DRG) and poles.

Periode : From january to mars

		Pôle 1	Pôle 2	Pôle 3	Total
DRG	TYPE DRG TITLE	288	318	519	1125
1	SURG CRANIOTOMY AGE >17 W CC	253	26	311	590
2	SURG CRANIOTOMY AGE >17 W/O CC	274	520	335	1129
3	SURG CRANIOTOMY AGE 0-17	225	319	212	756
4	SURG NO LONGER VALID	325	215	122	662
5	SURG NO LONGER VALID	125	138	118	381
Total		1490	1536	1617	4643

Figure 2: Example of a MT.

The DW presented in Fig. 1, offers several indicators to respond to users' needs (users from different profiles). In the context of the PMSI, we consider the following indicators:

1) *Offer indicators*: these indicators present the resources according to different dimension levels of a structure "structure", for instance:
   - The beds number of type "Medicine-Surgery-Obstetrics" to indicate the capacity to receive patients.
   - The main specialties by pole, to identify the types of diseases the hospital is able to treat.
2) *Needs and patient flow indicator (care consumption):* these indicators are mainly based either on the patient age or exit mode, for instance:
   - Describes the sojourn, analyze sojourns according to the group of diseases.
   - The main specialties of a pole.
   - Identify the population susceptible to be treated.
3) *Patient flow indicators:* these indicators presents the cause of the hospitalization and patient destination:
   - Where do the mothers come from?
   - What is the destination of the mother after the childbirth?

Various resources have been developed, to compute indicators from data, to analyze, visualize and aggregate data, elaborate dashboards and so forth. These existing exploitation resources are often numerous and of different type: formulas, OLAP requests, excel tables, and so on.

B. Aim of our research

Users have different profiles. In our context, for example, they belong either to the medical domain or to other domains. DW resources are numerous and complex, it is not easy for users from different domains to find relevant resources. In addition, existing resources do not have the same significance for these users from different profiles. In this context, we noticed many difficulties. We identify a *semantic lack related to DW concepts*: dimensions definition, measurements calculation methods and resources sources. Because of this semantic lack, the users cannot understand the usage of the DW resources that may respond to their needs. On the other hand, there is vocabulary *heterogeneity in query expression*: users do not belong to the same domain. They do not have the same vocabulary background. They do not express their need with the same terms (e.g., number of sojourn could be expressed as number of venue). Finally, concerning analysis needs, most of the time, users need to analyze many resources to make a decision. In big institutions, like the APHM, big numbers of resources make this task complicated. Thus, users need to have a global vision about resources responding to his requirements (e.g., calculation date, sources, criteria considered to calculate an indicator).

Consequently, to find, understand and choose relevant resources is a difficult task for users. Our challenge is to support users from heterogeneous domains in the exploitation of the existing resources. To this purpose, we propose to develop a personalization system supporting the users to exploit DW resources. We should note that our proposal is not limited to the healthcare domain. It can be used in other business contexts where users are from heterogeneous domains. In general, this is the case in big institutions.

The Ontology-driven Personalization System (OPS) is dedicated to support users from heterogeneous domains to exploit existing DW resources. This support is based on a knowledge base describing the domain (in this paper, we consider PMSI domain), the DW schema and the resources description. The following section presents the architecture of OPS and three scenarios of user support possibilities.

IV. An Ontology-Driven Approach for DW Personalization

The OPS supports users from heterogeneous domains to exploit existing DW resources. This support is based on a knowledge base that takes in consideration user domain, the DW schema and resources description. In order to provide such a personalization system, we developed an ontology-driven approach. In this section, we present first the general architecture of our ontology-driven personalization system, and then we present some uses-cases supported by OPS system.

A. General architecture

The general architecture of our OPS is illustrated in Fig. 3. OPS take in consideration information's collected from different DW construction levels. For example, at the conceptual level stores the DW schema.

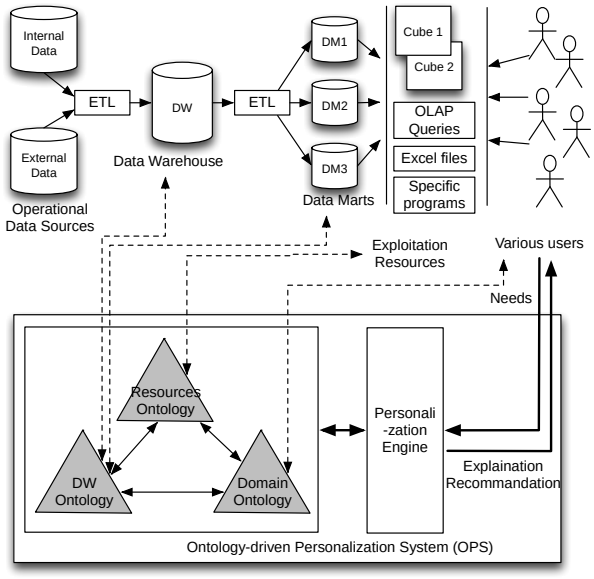


Figure 3: Ontology-driven Personalization System Architecture.

The two main components of our OPS are:

- *Knowledge base*: is an OWL database based on three related ontologies described in order: Domain (Hospital management - PMSI), DW schema (conceptual model), and existing DW resources.
- *Personalization Engine (PE)*: is the sub-system that personalizes users' interactions; the user expresses his needs to the OPS and the system provides semantic explanations or DW resources recommendations. This issue is based on the reasoning of the three ontologies.

B. Use cases of the Ontology-driven Personalization System

Several scenarios of user support functionality of OPS have been defined to develop and test the OPS. Each scenario corresponds to a user need expressed by a request addressed to the OPS (input). The OPS responds to the user with an explanation or a recommendation (output) depending on the nature of the expressed need. Examples of use-cases or expressed needs include:

1) ***Use-case 1:***

Entry: DW concept.

Output:

- *Domain concepts* - What are the existing measures to analyze a domain concept?
- *DW schema concepts* - What is the DW related concepts, measures: What are the different measures related to an analysis axe? What are the different analysis axes related to a measure? What are the measures that could be analyzed over a dimension?
- *Resources concept* - What are the existing resources to analyze a measure?

2) ***Use-case 2:***

Entry: Resources concept.

Output:

- *DW schema concepts* - What is the DW that provides a resource?
- *Domain concepts* - What are the existing resources to analyze a domain concept?
- *Resources concept* - What are the existing resources to analyze a measure?

3) ***Use-case 3:***

Entry: Domain concept.

Output:

- *Domain concepts* - What are the related domain concepts?
- *DW schema* – What are the domain concepts related to DW concepts?
- *Resources concept* - What are the resources related to a domain concept?

These use-cases are treated in the OPS by the PE reasoning on one or more ontologies. Fig. 4 illustrates the connection between ontologies and the users.

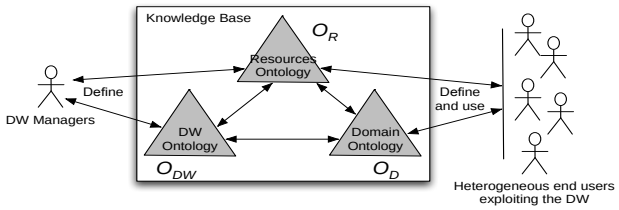


Figure 4: On the use of ontologies.

We distinguish two types of users:

- *The DW manager user*: he is in charge of the DW management and exploitation. He is mainly interested about the O_{DW} and the operational resources of the O_R.
- *The end-users*: they are heterogeneous; they search for resources that respond to their need. They expect resources and recommendations from the OPS to exploit the DW. These end-users express their needs using concepts belonging to the O_D and the conceptual resources, part of the O_R.

In this paper, we focus on the methodology used to develop the knowledge base composed of three ontologies: O_D, O_{DW} and O_R.

V. KNOWLEDGE BASE COMPONENTS

This section presents the Knowledge base of our OPS. This knowledge base is composed of three ontologies: O_D, O_{DW} and O_R. We present each of these three ontologies, the knowledge concerned, the methodology used to develop it and the models obtained in UML or in OWL.

To elaborate these ontologies, we use the ontology editor OWLGrEd. OWLGrEd uses a textual syntax OWL Manchester to create, edit and view an ontology [51]. OWLGrEd provides a comprehensive overview of OWL ontology with UML. OWLGrEd visualizes OWL classes as UML classes, data properties as attributes of classes, object properties as associations, individuals as objects and cardinality restrictions, associations between domain classes as UML cardinalities. To visualize other constructors of OWL, OWLGrEd enriched the UML class diagram with new notations [50], [51].

A. Domain ontology (O_D)

This sub-section present the description, the elaboration method and some exploitation results of the O_D.

1) Description:

The O_D gathers and streamlines the vocabulary related to a domain. Domain concepts are semantically related and defined in the ontology.

2) Elaboration methodology:

There are two solutions to obtain O_D. We can extract a part of existing O_D or create a new one manually. In the

first case, the ontology can be extracted from the existing ontology using ProSé plugin available with Protégé editor, it ensures the completeness of the extracted ontology [52]. As no O_D exists concerning "PMSI domain" we develop a new one.

To develop this O_D we decided to use UML, because this language is more user friendly for domain experts, and makes the validation process of the ontology easier. The methodology used to elaborate this ontology is illustrated in Fig. 5.

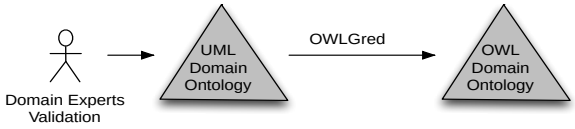


Figure 5: Domain Ontology Development.

Fig. 6 presents the O_D schema with UML. This schema is inspired from the model studied and presented in [53].

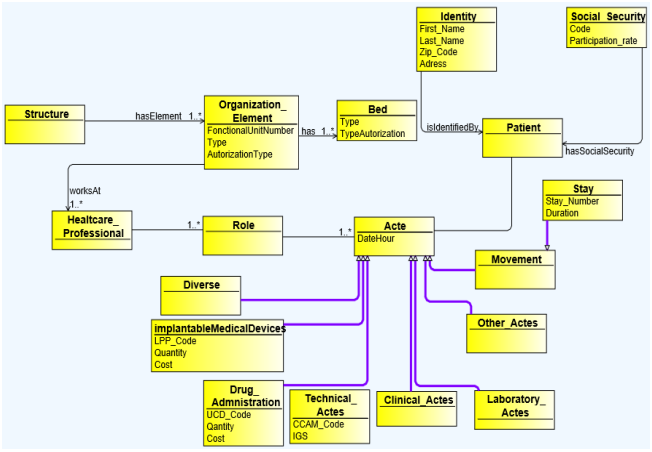


Figure 6: O_D schema presented with OWLGred.

This model is enriched and validated by domain experts. This ontology is presented here with the OWLGred tool.

3) Results:

The schema in Fig. 6 is used to validate the O_D with domain experts. Then, the UML schema is exported to OWL via the OWLGred tool. The O_D in OWL is visualized with Protégé in Fig. 7.

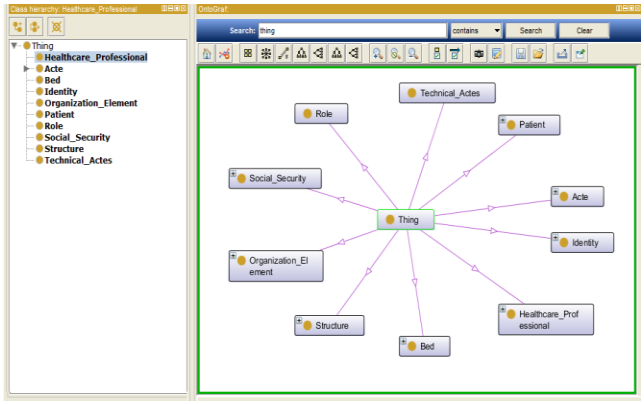


Figure 7: O_D presented with Protégé/OntoGraf.

This O_D is connected to other ontologies with semantic relations. O_D describes existing domains. OPS gives the possibility to visualize ontology concepts and relations between them, either they belong to the same ontology or not.

B. DW ontology (O_{DW})

This sub-section present the description, the elaboration method and some exploitation results of the O_{DW}.

1) Description:

Multidimensional model associated to the DW organizes data into facts and dimension. The O_{DW} concerns the DW conceptual schema. Facts represent the subject of analysis and dimensions represent the axes of analysis. Fact table is the center of the multidimensional model. It stores elementary indicators, called measures. Dimensions can form hierarchies, structured in different granularity levels.

2) Elaboration methodology:

To construct the O_{DW} we use a specific process. The first step of the process starts with the creation/extraction of the ROLAP structure of the DW (metabase) based on the SQL script of the relational data base of the DW. Then we annotate the tables with the multidimensional concepts (e.g., fact, dimension).

The atomization of this transformation from the conceptual model of the DW (the script SQL of the create table) to OWL is based on the research work of Prat *et al.* [54], Fig. 8 presents the O_{DW} development process. The research work of Prat *et al.* [54] defines a multidimensional meta-model, the concepts of OWL-DL, and transformation rules for mapping a multidimensional model into OWL-DL ontology.

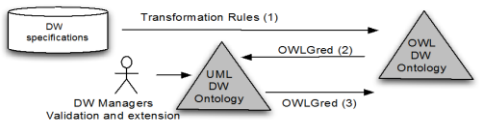


Figure 8: O_{DW} development process.

To generate the O_{DW} in OWL, the transformation rules proposed by Prat *et al.* [54] are adapted to our problematic. To validate and extend the model with DW manager the ontology is presented in UML. OWLGred tool translate the ontology from OWL script to UML. This process is illustrated in Fig. 8.

3) Results:

For the transformation of the O_{DW} from OWL intto UML we used OWLGred tool. Let's take the example of the DW schema Fig. 1, in the O_{DW} the dimension: D_Time is presented as a concept A_Time_Dimension. This concept have different dimension level Day, week, month and year. Those concept are presented with OWLGred (Fig. 9).

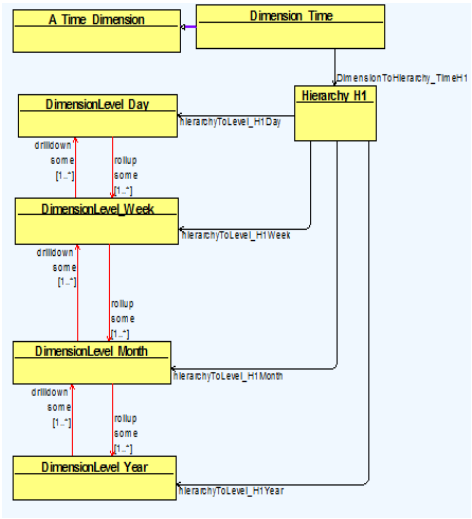


Figure 9: Dimension "D_Time" schema presented with OWLGred.

After the transformation of concepts from UML to OWL, O_D are visualized with Prtoégé/OntoGraf in Fig. 10.

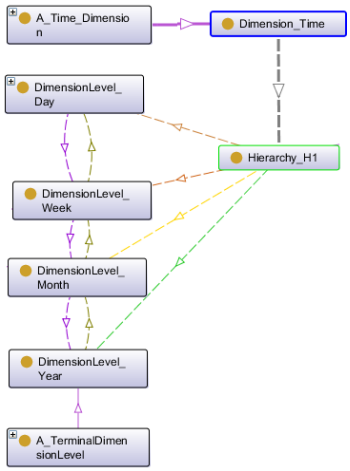


Figure 10: Dimension "D_Time' presented with Protégé/OntoGraf.

The O_{DW} is connected to other ontologies with semantic relations. This ontology presents the DW structure. It is mainly used by the DW manager.

C. Resources Ontology (O_R)

This sub-section present the description, the elaboration method and some exploitation results of the O_R.

1) Description:

Even if the multidimensional model is based on the metaphor of the cube or hypercube, the most common structure of the visualization is the MT presented in Fig. 2, which provides data presented in two axes of analysis [55], [3] enabling the visualization of a slice of the cube. Note that, other visualization possibilities exist to present the DW data (e.g., histograms, graphs).

Resources are related to the DW and are defined by the DW managers. To understand a resources components a user needs to have description information (e.g., calculation method, unit of measure, calculation period, date of creation, date of update, date of validity, objective, definition and the relation with the DW). We identified two types of DW resources:

- *Operational resources:* they concern the direct exploitation of DW, the resources requires an execution before being used for analysis (e.g., OLAP queries). They are used by the DW manager.
- *Conceptual resources:* they are user-oriented, they are resources used by the end-users (e.g., Excel files).

2) Elaboration methodology:

To develop O_R, as for O_D, we use UML for the same reasons. The conceptual resources (user-oriented resources) are validated by domain experts/users, and the DW managers validate the operational resources. The methodology used to elaborate this ontology is illustrated in Fig. 11.

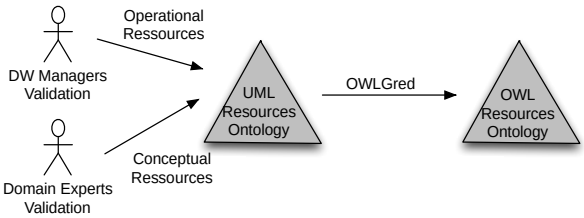


Figure 11: O_D development process.

Once the O_R expressed in UML class diagram, is validated with domain experts, it is transformed it into OWL with OWLGred tool (Fig. 12).

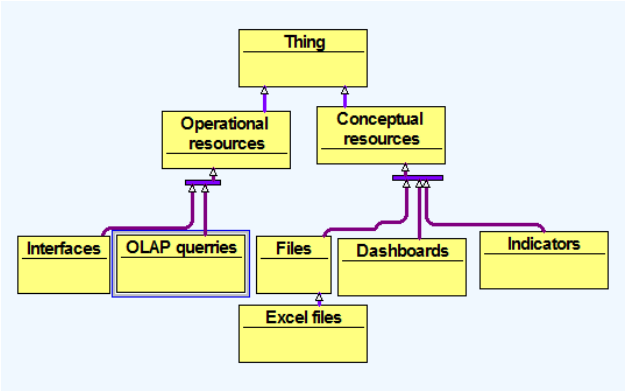


Figure 12: Extract of O_R schema, presented with OWLGRED.

3) Results:

The ontology O_R in OWL is visualized with Protégé tool in Fig. 13.

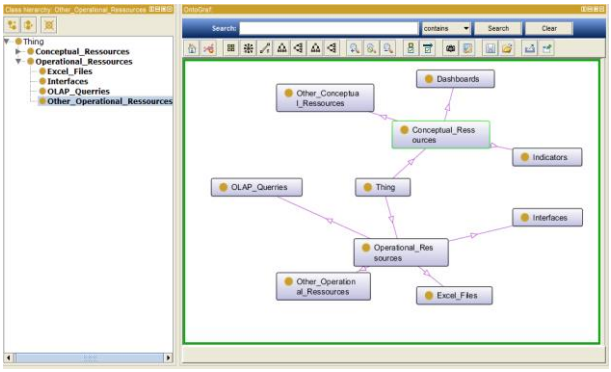


Figure 13: O_R presented with Protégé/OntoGraf .

This O_R is connected to other ontologies with semantic relations. This ontology enhances resources with descriptions. This ontology is mainly used by end-users.

VI. MAPPING ONTOLOGIES

The knowledge base of OPS is composed of three ontologies: O_D, O_{DW} and O_R. We formalize our ontology by the quadruple < O_{DW}, O_R, O_D, Map> where:

- O_D is the O_D that provides a schema about the domain.
- O_{DW} is a DW schema that describes DW schema.
- O_R is a resources ontology that describes the resources related to the DW.
- *Map* is the mapping between O_{DW}, O_R and O_D that establishes the connection between domain concepts, the DW and the resources components.

These mapped ontologies can be used for many purposes with OPS. On the one hand, to give a vision about the relation between DW resources and domain concepts, and on the other hand, to propose to users other related resources to make analysis based on reasoning technologies.

In this section, we focus on the *mapping* of these ontologies permitting this reasoning. We describe the mapping process. Then, we define the mappings between the three ontologies.

A. On the Mapping process

Considering two ontologies O_S and O_T, a *mapping* M between O_S and O_T, is a (declarative) specification of the semantic overlap between O_S and O_T at the concept level (Tbox). This *mapping* can be one-way (injective) or two-way (bijective). In an injective mapping, we specify how to express terms in O_S using terms from O_T in a way that is not easily invertible. A bijective mapping works both ways, i.e., a term in O_T is expressed using terms of O_S and the other way around. In ontology engineering, the following processes are pre-defined [56]:

1) *Ontology Merging* concerns ***creation of one new ontology from two or more ontologies***. In this case, the new ontology unifies and replaces the original ontologies. This often requires considerable adaptation and extension of the ontology.
2) *Ontology Aligning* brings the ontologies into mutual agreement. The ontologies are kept separate, but at ***least one of the original ontologies is adapted***, such as the conceptualization and the vocabulary match in overlapping parts of ontologies.
3) *Ontology Mapping (or relating ontology)* specifies how the concepts in different ontologies are related in a logical sense. This means that the original ontologies had not changed, but that ***additional axioms describe the connection between the concepts***. Leaving the original ontologies unchanged often implies *only a part of the integration*, because major differences may require adaptation of the ontologies.

As each of these ontologies can evolve, we do not choose the merging strategy to limit the impact of evolution changes. We prefer to keep three separate ontologies to limit the changes only to the connection (mapping) between them if necessary. Consequently, in our case, we have opt for *Ontology Aligning* or *Ontology Mapping* processes as defined before.

B. Concerned mappings

In our case, we considered three different mappings connecting these three ontologies two by two, depending on the connection between users and ontologies (Fig. 14).

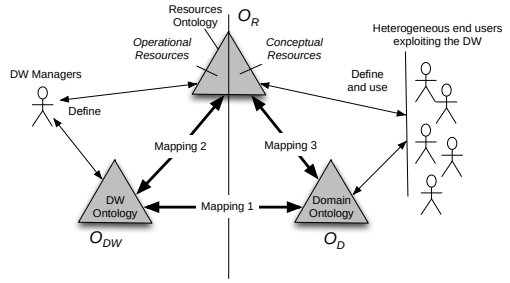


Figure 14: Different mappings between the three ontologies.

Mapping the three ontologies is necessary to facilitate the navigation between them. The Mapping 1 supports the connection between O_{DW} and O_D, Mapping 2 supports the connection between O_{DW} and the operational resources of the O_R, and, finally, Mapping 3 supports the connection between O_D and the conceptual DW resources of the O_R.

Ontology Aligning or *Mapping* processes related to these three mappings concerns: first searching similarities between ontologies, and then specifying mappings between ontologies. In our case, these two tasks are performed in a *manual manner using Protégé.*

1) Mapping 1: $O_{DW} – O_D$

This mapping is the first mapping to consider, because it is closely related to the DW design: a concept of the O_{DW} can be related to one or more concept(s) of the O_D, and one concept of the O_D can be related to one or more concept(s) of the O_{DW}.

2) Mapping 2: $O_{DW} – O_R$

This mapping can be considered as an extension of the O_{DW} towards operational resources of O_R: a concept of operational resource can be related to one or more concept(s) of O_{DW} (e.g., OLAP Query concept can be related to fact and dimension concepts). On the other side, a concept of the DW schema can be related to one or more concepts(s) of operational resources. For example, a measure can be implied in OLAP Query and Excel file. The O_{DW} concepts and O_R concepts concerned by this mapping are the lower classes of the respective ontology.

3) Mapping 3: $O_D – O_R$

Mapping 3 is deduced. The relation between O_D and O_R is identified through a process of deduction based on the transitive relation between O_D and O_{DW}. We present in Table I an example with OWL-DL.

TABLE I. CONCEPTS AND INFERRED CONCEPTS WITH OWL-DL.

Ontology	Concept
O_{DW}	A_Hospital_Structure_Dimension ⊆ A_Dimension
O_D	Structure
O_R	Resources1
O_{DW} - O_D	A_Hospital_Structure_Dimension ≡Structure
O_{DW} - O_R	Resources1ToDimension_Structure T ⊆ ∀Resources1ToDimension_Structure.Structure T ⊆ ∀Resources1ToDimension_Structure⁻.Resources1

This example presents the ontologies and their concepts "O_{DW} concepts", "O_D concept", "O_{DW} and O_D related concepts", "O_R concept" and finally "reasoning result concepts between O_D–O_R".

VII. VALIDATION EXAMPLE

To illustrate our proposal we suggest to respond to "Use-case 3" questions, we'll use OntoGraf [57] to visualize the ontologies' concepts. Fig. 15 shows the results of the search done on the mapped ontologies.

To show the definition and the concepts related of "DRG". The user enters "DRG".

Entry: Domain concept "DRG".

Output:

- *Domain concepts (from O_D):* the concept defining the 'Diagnosis related groups' (the user can access to the concept definition).
- DW schema element *(from O_{DW})*: the concept presenting a dimension 'D_DRG', note that D_DRG is a subclass of Dimension (Dimension ⊆ D_DRG).
- Resources concept *(from O_R)*: the concept identifying a resource 'Resource_Activity_Pole_DRG', this concept describes a multidimensional table representing data about PMSI activity per DRG and per Pole).

The benefits of a connected ontology is the information that it provides to describe a resource. The returned information is not only from O_R, it is also about connected concepts from O_D, and O_{DW}.

This section, presented preliminary test done by DW manager to define and validate the ontologies. However, end-user uses OPS system to search for resources that respond to his needs. OPS is based on O_D, O_{DW} and O_R connected ontologies to visualize the description of each resource.

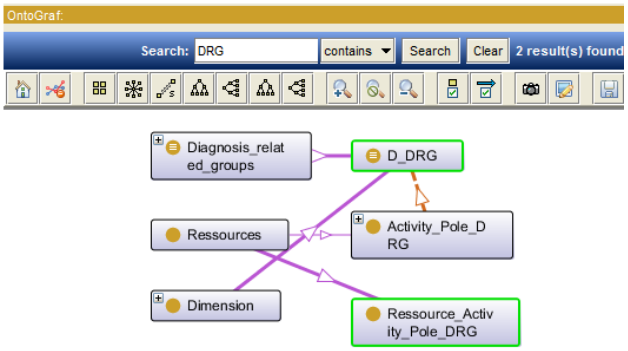


Figure 15: Example, retrieve "DRG" concept from the ontology Protégé/OntoGraf.

Thus, in the real application, OPS returns a resource with a set of information's form ontologies concepts describing the resource. For example, the resource

presented in Fig. 2 will be visualized with a set of descriptions, presented in Fig. 16.

DRG	TYPE DRG TITLE	Pôle 1	Pôle 2	Pôle 3	Total
1	SURG CRANIOTOMY AGE >17 W CC	288	318	519	1125
2	SURG CRANIOTOMY AGE >17 W/O CC	253	26	311	590
3	SURG CRANIOTOMY AGE 0-17	274	520	335	1129
4	SURG NO LONGER VALID	225	319	212	756
5	SURG NO LONGER VALID	325	215	122	662
	Total	125	138	118	381
		1490	1536	1617	4643

Objectif : number of patients per DIAGNOSIS, per POLE, per PERIOD.
Resource administrator: X Dupont.
Considered criterion: the considered diagnosis classification is the DRG used in the context of the PMSI, ...
Criterion defined by: Dr. Y Mouri, X Lalo.
Validity date : January to mars.
Update date: 01/04/2014.
Keywords: DRG, PMSI, Pole, Total, number of patients, period.
File name: number_of_patients_per_DRG_pole_period.xls

Figure 16: Example, resource with description from the three ontologies.

OPS have a user-friendly interfaces that offers several functionalities to end-users, for example, resources description or personalized resources retrieval.

VIII. Conclusion and Future Work

Ontologies are used in several domains to resolve syntactic and semantic heterogeneity problems. They facilitate the management of data, clarify and give a sense to ambiguous concepts. In a healthcare management context based on PMSI, numerous existing DW resources are provided to exploit a DW, they are shared by users from heterogeneous domains. These resources can be interpreted differently from a user to another. In addition, the personalization of specific and relevant resources to user is the aim of this research.

In this recent research field various studies propose different approaches to treat personalization problems, but they appear to be not adapted to our problematic. Indeed, the specificities of data related to healthcare management require semantic resources, in particular to tackle the heterogeneity of the users' profiles and domain complexity.

We have proposed an ontology-driven approach for a DW personalization system, in order to support heterogeneous users to explain or personalize (recommend) some existing DW resources adapted to their needs. This approach is based on a personalization engine using a knowledge base composed of three specific and related ontologies: O_D, O_{DW} and O_R.

In this paper, in progress of our proposition presented in [1], we focused on the elaboration of OPS knowledge base. We introduced the methodology used to develop each of the knowledge base ontologies, and presented the three ontologies models obtained in UML and in OWL languages. Then we have presented the mappings between these ontologies. To illustrate the use of this knowledge base to provide some resources explanations or recommendations to users, we have simulated the personalization engine using Protégé editor. We also queried and visualized ontologies with OntoGraf. We validated our approach by testing it on a simple user-case related to the healthcare domain, characterized by users' heterogeneity and domains complexity. We should note that our approach is not restricted to this domain; it can be applied in others domains.

This work leads to many other tasks. Future works on this research concern first the development of a user-friendly personalization engine of OPS, giving the user a friendly environment to query, provides resource explanation and resource personalization (recommendation). Then a validation process of the OPS has to be performed in a larger context, with DW managers' and end-users. Finally, we expect to study the impact of ontology evolution on OPS.

Acknowledgment

The authors wish to thank Sophie Rodier, Laetitia Malavolti, Bernard Guisiano, Jean-Francois Mirreti, Ghislaine Morvillez and Hélène Lherbet from Assistance Publique des Hôpitaux de Marseille (APHM), for their valuable contribution to this research.

Reference

[1] L. EL Sarraj, B. Espinasse, T. Libourel, and S. Rodier, "Towards ontology-driven approach for data warehouse analysis. Case study: healthcare domain," The Eighth International Conference on Software Engineering Advances (ICSEA 2013) IARIA, Oct. 2013, pp. 426-431, ISSN: 2308-4235, ISBN: 978-1-61208-304-9

[2] W. H. Inmon, Building The Data Warehouse, John Wiley & Sons, 1992.

[3] W. Lehner, "Modelling large scale OLAP scenarios," Advances in Database Technology (EDBT), 1998, pp. 153-167.

[4] A. Bosworth, J. Gray, A. Layman, and H. Pirahesh, "Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total," Data Min. Knowl. Discov., pp. 152-159, 1995.

[5] S. Rizzi, A. Abello, J. Lechtenborger, and J. Trujillo, "Research in data warehouse modeling and design: dead or alive?," The 9th International Workshop on Data warehousing and OLAP (DOLAP 2006) ACM, pp. 3-10.

[6] M. Golfarelli, "From user requirements to conceptual design in data warehouse design – a survey," Data Warehousing Design and Advanced Engineering Applications: Methods for Complex Construction, IGI Global, 2010.

[7] R. Kimball, and M. Ross, The Data Warehousing Toolkit, John Wiley&Sons, 1996.

[8] N. Prat, and J. Akoka, "From UML to ROLAP multidimensional databases using a pivot model," The 8th Journées Bases de Données Avancées, 2002, pp. 24.

[9] A. Tsois, N. Karayannidis, and T. K. Sellis, "Mac: Conceptual data modeling for OLAP," The 3rd International Workshop on Design and Management of Data Warehouses (DMDW 2001), Theodoratos 2001, pp. 2001.

[10] L. Bellatreche, D. Nguyen Xuan, G. Pierra and H. Dehainsala, "Contribution of ontology-based data modeling to automatic integration of electronic catalogues within

engineering databases," Computers in Industry Journal Elsevier, vol. 57, no. 8-9, pp. 711-724, 2006.

[11] L. Bellatreche, S. Khouri, I. Boukhari, and R. Bouchakri, "Using ontologies and requirements for constructing and optimizing data warehouses," Proc. of the 35th International Convention (MIPRO 2012), pp.1568-1573, May 2012.

[12] N. Prat, I. Megdiche, and J. Akoka, "Multidimensional models meet the semantic web: defining and reasoning on OWL-DL ontologies for OLAP," The 15th International Workshop on Data Warehousing and OLAP (DOLAP 2012) ACM, pp. 17-24, doi: 10.1145/2390045.2390049.

[13] S. Khouri, L. Bellatreche, and N. Berkani, "MODETL: a complete modeling and etl method for designing data warehouses from semantic databases," The International Conference on Management of Data (COMAD 2012), pp. 113.

[14] T. Niemi, and M. Niinimäki, "Ontologies and summarizability in OLAP," The 2010 ACM Symposium on Applied Computing (SAC 2010), pp. 1349-1353, doi:10.1145/1774088.1774378 http://doi.acm.org/10.1145/1774088.1774378.

[15] B. Neumayr, S. Anderlik, and M. Schrefl, "Towards ontology-based OLAP: datalog-based reasoning over multidimensional ontologies," The 15th International Workshop on Data warehousing and OLAP (DOLAP 2012), pp. 41-48, doi: 10.1145/2390045.2390053.

[16] Y. Ioannidis, and G. Koutrika, "Personalized systems: models and methods from an ir and db perspective," The 31st conference in the series of the Very Large Data Bases conferences (VLDB 2005), pp. 1365–1365.

[17] R. R. Korfhage, Information Storage And Retrieval, John Wiley & Sons, 1997.

[18] C. Domshlak, and T. Joachims, "Efficient and non-parametric reasoning over user preferences," User Modeling and User-Adapted Interaction, vol. 17, no. 1-2, pp. 41-69, 2007.

[19] A. Kobsa, "Generic user modeling systems," User Modeling and User-Adapted Interaction archive, Kluwer Academic Publishers Hingham, vol. 11, The Adaptive Web, pp. 49-63, 2001.

[20] L. Bellatreche, A. Giacometti, P. Marcel, H. Mouloudi, and D. Laurent, "A personalization framework for OLAP queries," The Eighth International Workshop on Data Warehousing and OLAP (DOLAP 2005), 2005, pp. 9-18.

[21] H. Jerbi, F. Ravat, O. Teste, and G. Zurfluh, "Management of context-aware preferences in multidimensional databases. ," the Third International Conference on Digital Information Management (ICDIM 2008), pp. 669-675.

[22] I. Garrigos, J. Pardillo, J.-N. Mazon, and J. Trujillo, "A conceptual modeling approach for OLAP personalization," in Conceptual Modeling-ER Verlag Berlin Heidelberg, 2009, pp. 401-414.

[23] F. Bentayeb, O. Boussaid, C. Favre, F. Ravat, and O. Teste, "Personnalisation dans les entrepôts de données: bilan et perspectives," The Fifth Journées sur les Entrepôt de Données et Analyse en ligne (EDA 2009), 2009, pp. 7-22.

[24] H. Jerbi, G. Pujolle, F. Ravat, and O. Teste, "Recommandation de requêtes dans les bases de données multidimensionnelles annotées," Revue des Sciences et Technologies de l'Information, Ingénierie des Systèmes d'Information, vol. 16, no. 1, pp. 133-138, 2011.

[25] D. Xin, J. Han, H. Cheng, and X. Li, A, "Answering top-k queries with multi-dimensional selections: The ranking cube approach," in VLDB, 2006, pp. 463-475.

[26] A. Giacometti, P. Marcel, and E. Negre, "A framework for recommending OLAP queries," Proc. The eleventh international workshop on Data warehousing and OLAP (DOLAP 2008) ACM, Pages 73-80, doi: 10.1145/1458432.1458446.

[27] A. Giacometti, P. Marcel, and E. Negre, "Recommending multidimensional queries," The 16th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2009), 2009, pp. 453-466.

[28] C. Sapia, "On modeling and predicting query behavior in OLAP systems," Proc. Workshop on Design and Management of Data Warehouses (DMDW), 1999, pp. 2.1-2.10.

[29] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, "Query recommendations for interactive database exploration," The 21st International Conference on Scientific and Statistical Database Management (SSDBM 2009), 2009, pp. 3-18.

[30] H. Jerbi, F. Ravat, O. Teste, and G. Zurfluh, "Applying recommendation technology in OLAP systems," The 11th International Conference on Enterprise Information Systems (ICEIS 2009), 2009, pp. 220-233.

[31] A. Giacometti, P. Marcel, E. Negre, and A. Soulet, "Query recommendations for OLAP discovery driven analysis," The 12th International Workshop on Data Warehousing and OLAP (DOLAP 2009) ACM, 2009, pp. 81-88, ISBN: 978-1-60558-801-8.

[32] E. Negre, Exploration Collaborative De Cubes De Données, François Rabelais Tours, France, 2009.

[33] C. Favre, F. Bentayeb, and O. Boussaid, "Evolution et personnalisation des analyses dans les entrepôts de données. Une approche orientée utilisateur," The 25th Informatique des Organisations et Systèmes d'Information et de Décision (INFORSID 2007), 2007, pp. 308-323.

[34] T. Thalhammer, M. Schrefl, and M. Mohania, "Active data warehouses: complementing OLAP with analysis rules," Data & Knowledge Engineering, vol. 39, pp. 241-269, 2001.

[35] A. Aybuke, and W. Claes, "Engineering and managing software requirements," Springer, 2005.

[36] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, "The Description logic handbook: theory, implementation, and applications," Cambridge University Press, 2003, ISBN:0-521-78176-0

[37] T. Gruber, "A translation approach to portable ontology specification," Knowledge Acquisition, vol. 5, no. 2, pp. 199-220, 1993.

[38] N. Guarino, and P. Giaretta, "Ontologies and knowledge bases, towards a terminological clarification," IOS Press, Amsterdam, pp. 25-32, 1995.

[39] S. Jean, G. Pierra, and Y. Ait-Ameur, "Domain ontologies: a database-oriented analysis," Springer, vol. 1, pp. 238-254, 2007.

[40] G. Antoniou, and F. V. Harmelen, "Web ontology language: OWL," Handbook on ontologies, 2009.

[41] Object Management Group, "OMG formal," http://www.omg.org/spec/, Retrieved 2014.

[42] N. Berkani, S. Khouri, and L. Bellatreche, "Generic methodology for semantic data warehouse design: From schema definition to etl," in the Intelligent Networking and Collaborative Systems (INCoS2012), Sept. 2012, pp. 404-411, ISBN: 978-1-4673-2279-9

[43] "Fact++," http://owl.man.ac.uk/factplusplus/, Retrieved 2014.

[44] F. Arvidsson, and A. Flycht-Eriksson, "Ontologies I," 2008.

[45] G. Pierra, "Chapter context representation in domain ontologies and its use for semantic integration of data,"

Journal on Data Semantics X, vol. 4900, pp. 174–211, 2008.

[46] P. Coret, J. Richard, E. Talavet, and T. Trofimoya, Introduction A OWL, Langage XML D'Ontologies, Technical report, 2006.

[47] S. Khouri, L. EL Sarraj, L. Bellatreche, B. Espinasse, N. Berkani, S. Rodier, and T. Libourel, "CiDHouse: contextual semantic data warehouses," The 24th Database and Expert Systems Applications (DEXA 2013), Springer Berlin Heidelberg, 2013, pp. 458-465.

[48] F. Pinet, C. Roussey, T. Brun, and F. Vigier, "The use of UML as a tool for the formalisation of standards and the design of ontologies in agriculture," Advances in Modeling Agricultural Systems, vol. 25, pp. 131-147, 2009.

[49] R. Liepins, K. Cerans, and A. Sprogis, "Visualizing and editing ontology fragments with OWLGrEd," Proc. CEUR Workshop Proceedings, vol. 932, pp. 22-25, 2012.

[50] J. Barzdins, G. Barzdins, K. Cerans, R. Liepins, and A. Sprogis, "OWLGrEd: a UML style graphical notation and editor for OWL 2," Proc. CEUR Workshop Proceedings vol. 614, 2010.

[51] J. Barzdins, K. Cerans, R. Liepins, and A. Sprogis, "UML style graphical notation and editor for OWL 2," Perspectives in Business Informatics Research, Springer Berlin Heidelberg, vol. 64, pp. 102-113, ISBN:978-3-642-16100-1

[52] B. C. Grau, I. Horrocks, and Y. K. yevgeny, "Modular reuse of ontologies: theory and practice," Journal of Artificial Intelligence Research, 2008.

[53] GMSIH, "Tarification à l'activité (T2A) - Guide à l'usage des établissements de santé," Appui santé et médico-social (ANAP), 2005.

[54] N. Prat, J. Akoka, and I. Comyn-Wattiau, "Transforming multidimensional models into OWL-DL ontologies," The IEEE 8th International Conference on Research Challenges in Information Science (RCIS 2011) IEEE, 2011.

[55] M. Gyssens, and L. V. S. Lakshmanan, "A foundation for multi-dimensional databases," Proceeding the 23rd International Conference on Very Large Data Bases (VLDB 1997), pp. 106–115.

[56] J. D. Bruijn, F. Martın-Recuerda, D. Manov, and M. Ehrig, "D4.2.1 State-of-the-art survey on ontology merging and aligning V1," Semantically Enabled Knowledge Technologies (SEKT 2003), 2003.

[57] S. Falconer, "OntoGraf," In Protégé Wiki, from http://protegewiki.stanford.edu/wiki/OntoGraf, Retrieved 2014.

Formal Models in Software Development and Deployment: A Case Study

Radek Kočí* and Vladimír Janoušek†
Brno University of Technology, Faculty of Information Technology,
IT4Innovations Centre of Excellence
Bozetechova 2, 612 66 Brno, Czech Republic
*koci@fit.vutbr.cz
†janousek@fit.vutbr.cz

Abstract—**Modeling, implementation, and testing are integral parts of system development process. Models usually serve for description of system architecture and behavior and are automatically or manually transformed into executable models or code in a programming language. Tests can be performed on implemented code or executable models; it depends on used design methodology. Although models can be transformed, the designer has to usually adapt resulted code manually. It can result in an inconsistency among design models and their realization and the further development, testing and debugging by means of prime models is impossible. This work summarizes the design methodology based on the formalism of Object Oriented Petri Nets combined with Discrete Event System Specification and demonstrates its usage in the system development and deployment on the simple robotic system case study. The goal is to use the same formalisms for system modeling as well as for system implementation, so that to keep designed models in the deployed system.**

Keywords–Object Oriented Petri Nets, Discrete Event System Specification, multi-paradigm modeling, model deployment.

I. Introduction

This work is based on the paper [1], which is extended of detailed explanation of the design methodology and its usage for system development and deployment. It is demonstrated on simple, but fully described, case study.

Modeling, implementation, and testing are integral parts of system development process. Various models are used in analysis and design phases and usually serve as a system documentation rather than real models of the system under development. The system is then implemented according to these models, whereas the code is either generated from models or is implemented manually. Unfortunately, implementations often differ from the models because of debugging or system improvement. Consequently, models become out of date and useless.

To solve a problem with manual implementation and impossibility to test designed system using models, the methodologies and approaches commonly known as Model-Driven Software Development are investigated and developed for many years [2], [3] These methods use executable models, e.g., Executable UML [4] in Model Driven Architecture methodology [5], which allows to test systems using models. Models are transformed into another models and, finally, to code. Nevertheless, the resulted code has to often be finalized manually and the problem with semantic mistakes or imprecision between models and transformed code remains unchanged.

The approach to system development, which is presented in the paper, uses formal models as a means for system description as well as system implementation. The basic idea is to have a framework allowing to execute models in different modes, whereas each mode is advisable for another kind of usage—design, testing, and deployment. The system is developed using different kinds of models (from formal models to direct code in a programming language) in simulation, i.e., it is possible to test systems in any state in any time. The design method, which is taken into account in the papers [6], [7], does not require model transformations and assumes that models serve for system description as well as system implementation. The formalism of Object-Oriented Petri Nets (OOPN) [8], [9] and Discrete Event System Specification (DEVS) are basic modeling means.

The paper is organized as follows. First, we will attend to related work in Section II. The formalism of OOPN will be briefly introduced in Section III. Basic principles of modeling methodology will be described in Section IV, different modeling means will be compared in Section V, the approach to model system behavior will be presented in Section VI, and Section VII pays an attention to the architecture modeling. Finally, possibilities to deploy models into product environment will be discussed in Section VIII and Section IX concludes the paper and describes a future work.

II. Related Work

Combination of formal models, simulation, and model deployment is applicable mainly in control software. The use of high-level languages, especially Petri Nets, allows to build and maintain control systems in a quite fast and intuitive way. To control robot application, hierarchical binary Petri nets are used for middleware implementation in a RoboGraph framework [10]. To develop control software for embedded systems, the work that uses Timed Petri Nets for the synthesis of control software by generating C-code [11], the work based on Sequential Function Charts [12], or the work based on the formalism of nets-within-nets (NwN) [13], [14], [15] can be mentioned.

These tools and works allow to *model* systems using a combination of different formalisms, but do not allow to use formal models in system *implementation*. The proposed

approach allows to use formal models as a basic design, analysis and programming means combining simulated and real components. The main advantages; there is no need for code generation, and for further investigation of deployed systems, using the same formal models and methods is possible.

III. Formalism of Object Oriented Petri Nets

We will briefly introduce the formalisms of Object-Oriented Petri Nets. Object orientation of Object-Oriented Petri nets (OOPN) [16] is based on the well-known class-based approach. All objects are instances of classes, every computation is realized by message sending, and variables contain references to objects. This kind of object-orientation is enriched by concurrency. OOPN objects offer reentrant services to other objects and, at the same time, they can perform their own independent activities. The services provided by the objects as well as the autonomous activities of the objects are described by means of high-level Petri nets—services by *method nets*, object activities by *object nets*.

The formalism of OOPN contains important elements allowing for testing object state (*predicates*) and manipulation with object state with no need to instantiate nets (*synchronous ports*). Object state testing can be negative (*negative predicates*) or positive (*synchronous ports*).

An example illustrating the important elements of the OOPN formalism is shown in Figure 1. There are depicted two classes `C0` and `C1`. The object net of the class `C0` consists of places `p1` and `p2` and one transition `t1`. The object net of the class `C1` is empty. The class `C0` has a method `init:`, a synchronous port `get:`, and a negative predicate `empty`. The class `C1` has a method `doFor:`.

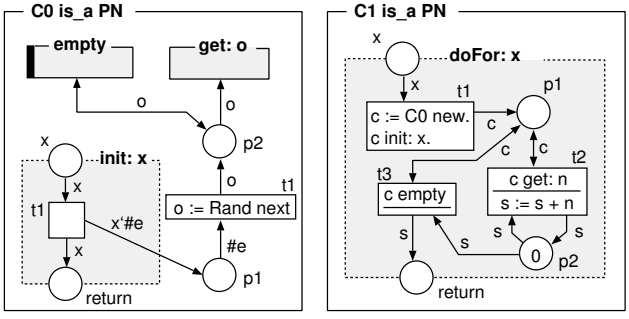


Figure 1. An OOPN example.

The OOPN dynamics is based on high-level Petri net dynamics, but the semantics of a transition is little bit modified. A transition is *fireable* for some binding of variables, which are present in the arc expressions of its input arcs and in its guard expression, if there are enough tokens in the input places with respect to the values of input arc expressions and if the guard expression for the given binding evaluates to true.

Synchronous ports are special (virtual) transitions, which cannot fire alone but only dynamically fused to some other transitions, which activate them from their guards via message sending. Every synchronous port embodies a set of conditions, preconditions, and postconditions over places of the appropriate object net, and further a guard, and a set of parameters. Parameters of an activated port $sync$ can be bound to constants or unified with variables defined on the level of the transition or port that activated the port $sync$. An example is shown in Figure 1—the port `get:` (class `C0`) having one formal parameter `o` is called from the transition `t2` (class `C1`) with free variable `n`—it means that the variable `n` will be unified with the content of the place `p2` (class `C0`).

Negative predicates are special variants of synchronous ports. Its semantics is inverted—the calling transition is fireable if the negative predicate is not fireable. The passed variable cannot be unbound (the unification is impossible) and the predicate cannot have a side effect. An example is shown in Figure 1, the predicate `empty` (class `C0`). This predicate is called from the transition `t3` (class `C1`)—it means that the transition `t3` will be fireable if the place `p2` (class `C0`) is be empty.

Let us investigate what happens after calling the method `doFor:` with a value `3` on an instance of the class `C1` (the instance will be denoted by obj_{C1}). First, the transition `t1` is fired with following actions: the instance of the class `C0` is created (the instance will be denoted by obj_{C0} and the reference to this object is assigned to the variable `c`) and initialized by the method net `init:`. It puts the symbol `#e` to the place `p1` of the object net obj_{C0} three times. The transition `t1` of the object net obj_{C0} generates three random numbers and puts them into the place `p2`. Second, the transition `t2` of the object net obj_{C1} tests if there is any object (a value) in the object net obj_{C0} by testing the synchronous port `get:`. If its evaluation is true, the transition `t2` is firable. If the transition `t2` fires, the synchronous port `get:` fires too. Since the variable `n` is free, the variable `n` is unified with a random number from the place `p2` of the object net obj_{C0}. The transition `t2` of the object net obj_{C1} then adds this value to the sum (the variable `s`). Otherwise, the the transition `t3` of the object net obj_{C1} tests if there is no value in the object net obj_{C0}—then the negative predicate `empty` is firable. If the transition `t3` fires, it places the sum (the variable `s`) to the return place as a method result. So, an invocation of the method `doFor:` leads to random generation of `x` numbers and to return of their sum.

IV. Modeling Methodology Principles

This section introduces modeling methodology, which has been presented by Kočí and Janoušek [17], and simple case study. Only basic principles of methodology will be shown here; details and the complex model of proposed case study will be being developed in Sections VI and VII.

A. Modeling Process

The modeling process is split up into three basic phases—identification of model elements, modeling the system architecture, and modeling the system behavior. Different modeling means are used in different steps, nevertheless, theses means are linked together. Brief description of basic phases and used modeling means follows:

- *Basic model elements* are *users* of the system, their *roles*, and *activities* of the system. To identify them, the *use case diagram* from UML can be used. Roles

are modeled by means of actors and activities by means of use cases.

- *System architecture* is modeled by means of class diagrams from UML. Modeled classes are linked to elements of use case diagrams as follows:
  - roles are represented by a group of classes modeling logic view at roles behavior,
  - activities are represented by a group of classes modeling functionality of the system,
  - users are represented by a group of classes modeling data of roles and accessing potentially present communication channels.
- *System behavior*. Behavior of roles and activities is modeled by means of Object oriented Petri nets formalism. It can also be modeled by any other formalism allowing to define workflow scenarios and offering an interface for workflows synchronization, e.g., statecharts, activity diagrams, or other kind of Petri nets. The comparison of chosen formalisms is done in Section V.

B. Layered Architecture of the Modeling Process

First, the relationship between *user* and *role* has to be explained. The role defines a work context the user can act in. For instance, the user working with a conference system can act as an author, a reviewer, a chair, or a combination of these roles. Thus, users act through their roles in the system and each user can have more roles. Nevertheless, the system can have other objects than users, that have the same or similar meaning, that is to represent data base, either for some of present roles or for data needed to be stored in the system. We denote such objects by a notion *subjects*.

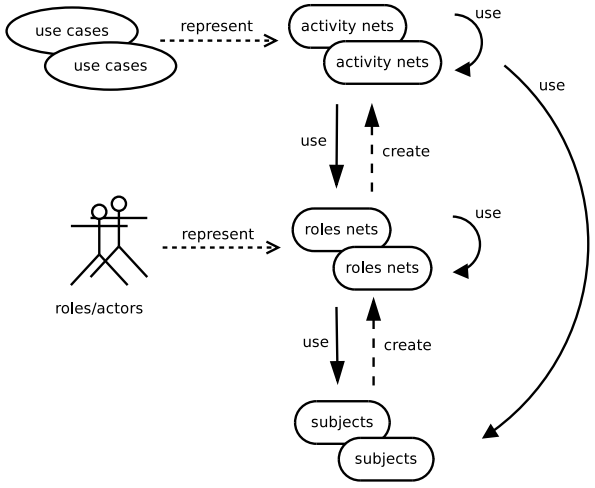


Figure 2. Design Method – Layers in the model.

Figure 2 shows model elements and their relationships in the design process. The design method distiguishes following model elements:

- *Subjects* represent a base for data storage or accessing communication channels.
- *Role nets* are derived from *actors* modeled in use case diagrams. Role nets model logic view at roles in the system and offer the communication protocol to the subject depending on the role intention.
- *Activity nets* are derived from use cases and model system functionality. The model is based on workflow definition and uses roles and subjects.

Model elements are organized in layers. Relationshisps are depicted by arrows—the solid arrow from *sender* to *receiver* represents a relationship a *sender* uses a *receiver*. For instance, activity nets can use roles nets, subjects, but also other activity nets. The dashed arrow from *sender* to *receiver* represents a relationship a *sender* creates a *receiver*. For instance, activity nets are created by role nets.

C. Simple Case Study

We will demonstrate basic principles of the design method on the example (simple case study) of a robot control system. We have a system with robots, where a motion of each robot is controlled by the same scenario, which is described by the following algoritm: (1) the robot is walking; (2) if the robot comes upon to an obstacle, it stops, turns right and tries to walk; (3) if the robot cannot walk, it turns round and tries to walk; (4) if the robot cannot walk, it stops. User can start and stop this scenario anytime.

V. UML and OOPN in the Design Process

As it was mentioned, the different formalisms or means can be used to describe system behavior. The comparison of using chosed UML models and formalism of OOPN will be presented in this section. It will be demonstrated on the previously instroduced case study.

First of all, the designer would identify model elements using use case diagrams. After analysing the case study specification, we can model use cases as shown in Figure 3. There we can find an actor *User* and a use case *Execute Scenario* representing the control algorithm.

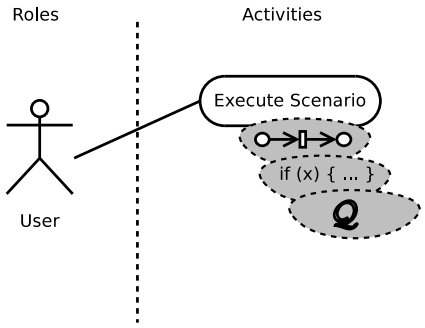


Figure 3. First Use case diagram of designed system.

A. UML in the Design Method

In UML, the use case specification is usually described informally by means of pure text or semi-structured text displayed in a table. Figure 4 shows such a table, which specify a behavior of the use case *Execute Scenario*. The table describes an algorithm in an informal way using keywords for text structuring, e.g., *IF–ELSE* branching, *REPEAT* a step, etc.

Name	Execute Scenario
Sequence of steps	
1. The scenario starts by user's stimulus 2. IF there is a clear road 2.1. the robot goes straight 2.1. repeat step 2 3. ELSE 3.1. the robot turns right 3.2. IF there is not clear road 3.2.1. the robot turns round 3.2.2. IF there is a clear road 3.2.2.1. REPEAT step 2	
Alternative sequence of steps	
1. User can stop this scenario anytime	

Figure 4. Specification of the Use Case *Execute Scenario*.

Another way to describe use case specifications is to use digrams from UML such as *activity diagram* or *statecharts*. Their usage allows for more precise description of behavior, which is based on predefined elements with clear semantics.

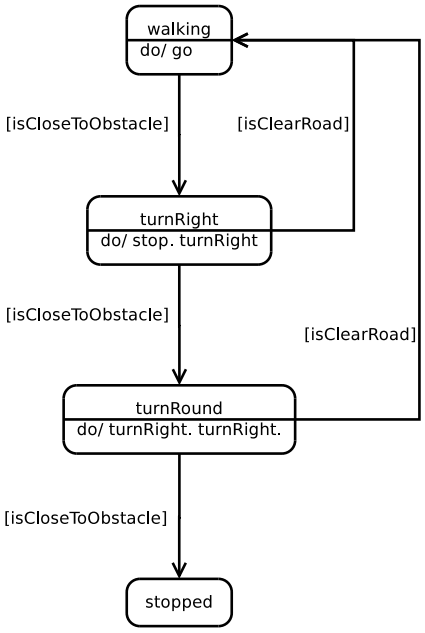


Figure 5. Statechart of the use case *Execute Scenario*.

The statechart modeling the use case *Execute Scenario* is shown in Figure 5. Four different states of modeled activity have been identified:

- *walking* means the robot walks
- *turnRight* means the robot turns right for the first time it came at an obstacle
- *turnRound* means the robot turns round for the second time it cannot go on
- *stopped* means the robot cannot go on

Each state containts commands that are executed in the state—*go* instructs the robot to go straight, *stop* instructs the robot to stop, and *turnRight* instructs the robot to turn right. If the command *turnRight* is sent twice, it means that the robot turns round.

Transitions between states are modeled by means of arcs whereas each transition can be conditioned. In the example, there are two conditions for transitions:

- *isCloseToObstacle* means the robot came at an obstacle and cannot go on
- *isClearRoad* means the robot can go on

B. Object Oriented Petri Nets in the Design Method

Methods that have been presented in Section V-A allow for use case description but their validation can be problematic because of impossibility to check models either by formal means or by simulation. Of course, there are tools and methods [4], [5] that allow to simulate modified UML diagrams. Nevertheless, there is still a strict border between *design* and *implementation*. On the other hand, if the formalisms allowing *to design* as well as *to implement* the system is used, we needn't care about borders and problems that can arise during a transition from design to implementation, and vice versa. One of such formalisms, which can be used to model use case (activity) behavior is the formalism of Object-Oriented Petri Nets (OOPN).

Let us continue in the example of *Execute Scenario*. The activity net *ExecuteScenario* of the use case *Execute Scenario*, which is described using OOPN, is shown in Figure 6.

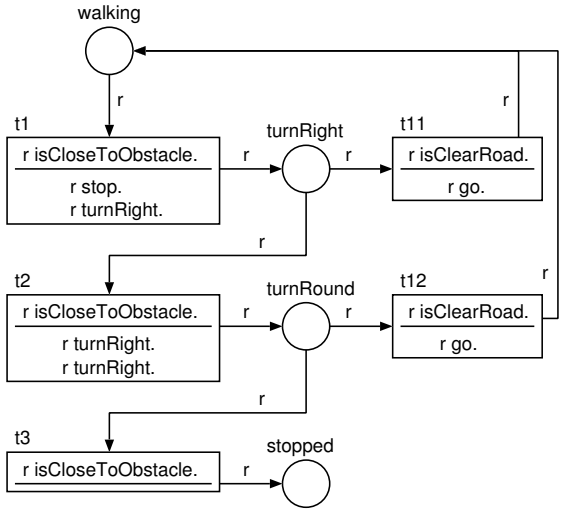


Figure 6. Activity Net *ExecuteScenario*.

The activity net contains elements similar to the statechart model shown in Figure 5. States are modeled by places `walking`, `turnRight`, `turnRound`, and `stopped` with the same meaning. Nevertheless, states `turnRight` and `turnRound` are only temporal and the activity goes through these ones to the one of stable states `walking` or `stopped`.

The control algorithm is represented by a sequence of transitions whereas each transition is conditioned by an event representing a change on robot's state. For instance, the transition `t1` has a condition *isCloseToObstacle* testing if

the robot is close to obstacle. If this condition is evaluated true, the transition `t1` is fired and appropriate commands are performed—*stop* and *turnRight* the robot. This, the transition `t1` models a behavior described in point 3.1 in Figure 4.

A short remark to OOPN notation. To record a sequence of transitions (i.e., events that happened in one scenario), we will type `<tName1,tName2,...>`. For instance, `<t1,t2,t3>` means, that the robot came at an obstacle and there is no possibility to go—the robot stops.

VI. System Behavior Modeling

As it was mentioned in Section IV, the modeling process is split up into three basic phases of identification of model elements, modeling the system architecture, and modeling the system behavior. The system behavior modeling will be presented in this section. It will be demonstrated on the previously instroduced case study.

A. Modeling phases cohesion

The important feature is the modeling phases cohesion. Phases are not separated but should be being provided simultaneously. It means, that model elements are finding and improving continuously.

If we analyze the activity net *ExecuteScenario* shown in Figure 6, we can see, that *isCloseToObstacle*, *isClearRoad*, *stop*, *go*, and *turnRight* are commands. Of course, these commands have to have their receiver communicating with a real (or simulated) robot—so, by specification of use case behavior, we have identified a new role in the system—the *Robot*. It also implies that the activity has to be linked to a role in the system—this role is stored in places and serves even as a state token. In our example, the role supplies an information about the robot and allows to send commands to it.

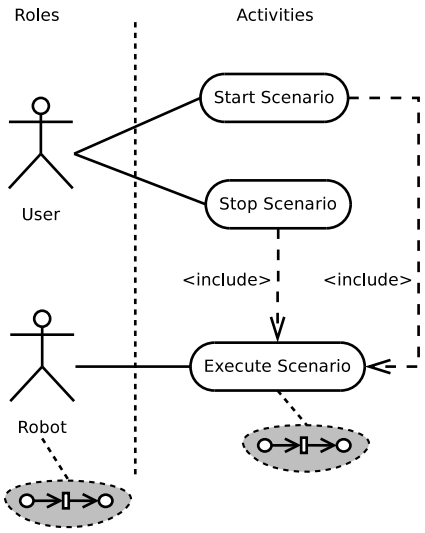


Figure 7. Use Cases of designed system.

So far, we did not take a care about two parts of the scenario—start and stop the robot (see points 1 of basic sequence and point 1 of alternative sequence in Figure 4). Because the robot behavior is an autonomous activity, and use cases we can make a decision to model start and stop as two separated activities. The updated model of use cases are shown in Figure 7. It introduces new role *Robot* and new use cases *Start Scenario* and *Stop Scenario*.

B. Activity Nets

We have presented the activity net *ExecuteScenario* (see Figure 6). The robot can be in two stable states—walking or stopped (there is no possibility to walk). Each such a state is represented by appropriate place, i.e., places `walking` and `stopped`. We have to be able to test activity states, therefore the predicates are generated for each such a place—the synchronous port `isStopped` and the negative predicate `isNotStopped` for the state `stopped` and similar predicates for the state `walking`. Test predicates are shown in Figure 8.

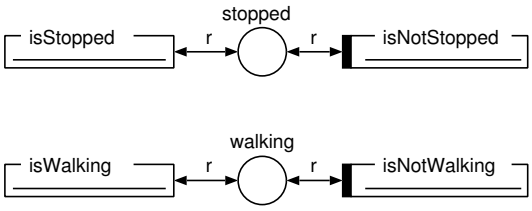


Figure 8. Activity Net *Scenario* – predicates.

Uses cases *Start Scenario* and *Stop Scenario* have to be modeled too. These use cases can be modeled in two ways—in a special activity net or in a method of existing activity net. Because these use cases work only with the activity net *ExecuteScenario*, how deduced from use case diagram in Figure 7, we can model them as methods. So, we have two activities that are using onother activity what is consistent with the layered architecture (Figure 2).

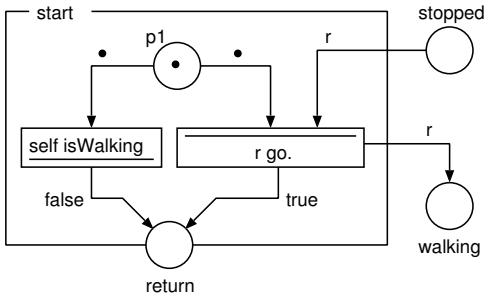


Figure 9. Activity Net *ExecuteScenario* – a method net *start*.

The use case *Start Scenario* is modeled by method net `start` shown in Figure 9. A decision what has to be done is based on the activity state—if the state is *walking* (tested by synchronous port `isWalking`), the method does nothing; if the state is *stopped*, it starts the robot's walk, i.e., sends a message `go` and moves the state token *role* from the place `stopped` to the place `walking`. The state *stopped* is not tested by a predicate, but the transition is directly conditioned by the place `stopped` because it will process the state token `role` in the case of success.

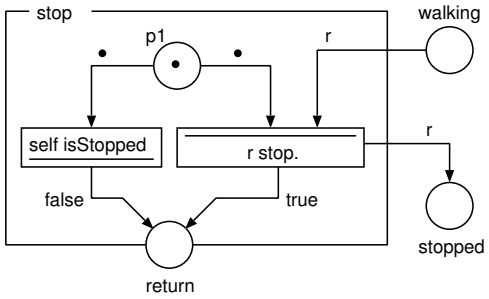


Figure 10. Activity Net *ExecuteScenario* – a method net *stop*.

The use case *Stop Scenario* is modeled by method net `stop` shown in Figure 10. It is similar to the method `start` having following differences. If the state is *stopped*, it does nothing. If the state is *walking*, it sends a message `go` and moves the state token *role* from the place `walking` to the place `stopped`.

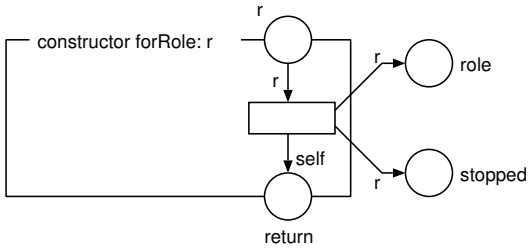


Figure 11. Activity Net *Scenario* – the constructor.

Each activity net is instantiated for just one role, so that the role is initialized by means of constructor as shown in Figure 11. The constructor has one parameter—the role the activity net is assigned to. So each activity has a place storing the role object; the presented constructor has the same structure for every activity nets. In addition, the constructor initializes activity—in our example, it puts a state token *role* to the place *stopped*. After initialization, the robot is in the state `stopped` (the robot stops).

C. Role Nets

A possible model of the role *Robot* is shown in Figure 12. The role checks actual distance of robot to an obstacle each 10 time units (the transition `t1`) and offers information about robot's position by means of predicates *isClearRoad* and *isCloseToObstacle*. To get information about the distance, the role asks its subject by sending a message *getDistance* (the transition `t2`).

Much like for the activity nets, each role net is initialized by means of the constructor having one parameter—the subject the role net is assigned to. Each activity has a place `subject` storing the subject. The constructor is shown in Figure 13 including the method `turnRight`—it only delegates the message to the subject. Other methods (*go* and *stop*) are modeled in similar way and are not shown.

Each role has its own set of activities it can participate in. The activity nets should be created by asking roles, not

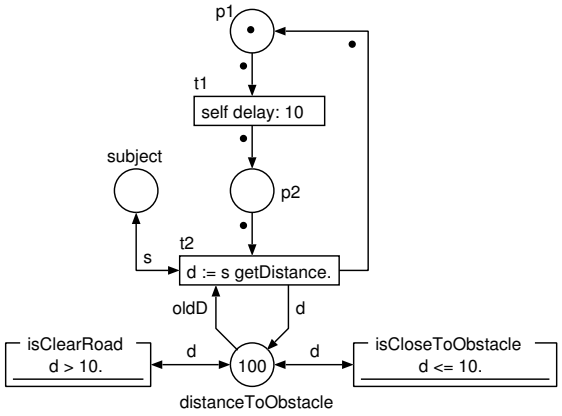


Figure 12. Object net of the role *Robot*.

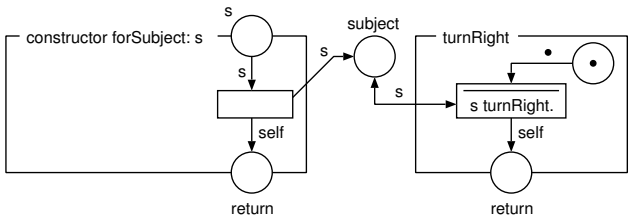


Figure 13. Methods of the role *Robot*.

directly. For instance, the role *Robot* has only one activity net *ExecuteScenario*, so that there is the method *createActivity*, which creates a new instance of activity net *ExecuteScenario*. The method is shown in Figure 14.

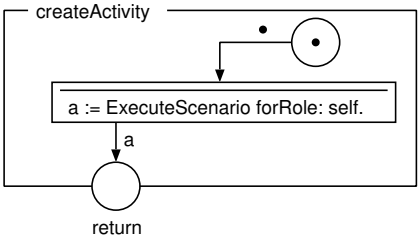


Figure 14. Activity creation of the role *Robot*.

D. Subject Models

Each role needs to have its subject, i.e., the object defining information about a subject, which can have different roles in the system. The subject is usually modeled as an object containing efficient data directly or as an interface to a database, another system or remote object. The subject can access the real system or can simulate the real system for the testing reason. The subject can be described by the same formalism as activity nets (i.e., by OOPN) or implemented in any language present in the product environment.

It this section, we will demonstrate usign OOPN for subject modeling. The subject for the role *Robot* is named *RobotDevice* and is shown in Figure 15. It represents simulated interface to the real robot in the system. The current distance to

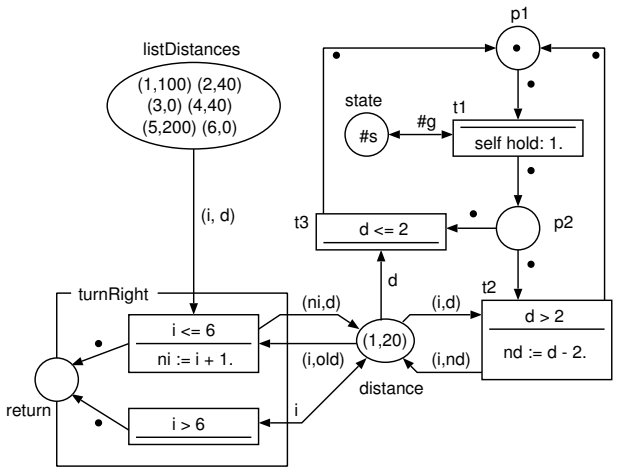


Figure 15. The subject *RobotDevice* – implementation with OOPN.

an obstacle is stored in the place `distance`. If the robot is instructed to go (i.e., the symbol `#g` is stored in the place `state`), the distance is decreased each one time clock (a sequence `<t1,t2>`). If the distance is less than 2 (it simulates the robot is by an obstacle), the distance does not change (a sequence `<t1,t3>`).

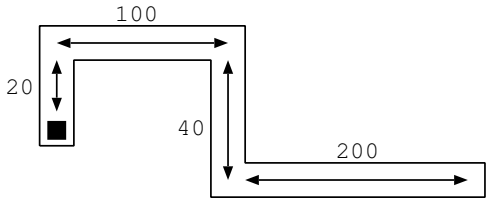


Figure 16. Labyrinth simulated by the subject *RobotDevice*.

The simulated labyrinth the robot is moving in is shown in Figure 16. Simulated robot device contains a list of distances to obstacles (the place *listDistances*), which are subsequently set if the method *turnRight* is performed. First distance is `20`, then it subsequently changes to following values:

- `100` – an item `(1,100)`; the robot turns right, there is a free road 100 units long
- `40` – an item `(2,40)`; the robot turns right, there is a free road 40 units long
- `0` – an item `(3,0)`; the robot turns right, there is an obstacle (a wall)
- `40` – an item `(4,40)`; the robot turns right, there is a road the robot went through (40 units long); in fact, this particular information will not be taken into account, because the previous distance was `0` and the activity net *ExecuteScenario* performs an operation *turn round*, which is implemented by calling *turnRight* twice—the item `(4,40)` will be skipped (see the activity net *ExecuteScenario* in Section V-B)
- `200` – an item `(5,200)`; the robot turns right, there is a free road 200 units long
- `0` – an item `(6,0)`; the robot turns right, there is an obstacle; any other actions invoke no changes

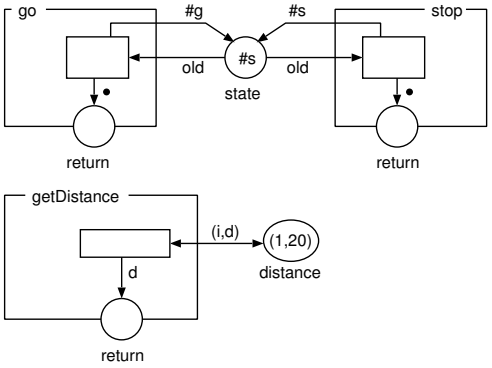


Figure 17. The subject *RobotDevice* – implementation with OOPN.

Methods `go`, `stop`, and `getDistance` is shown in Figure 17. The method `go`, respectively `stop`, puts a symbol `#g`, respectively `#s`, to the place `state`. The method `getDistance` gets a value from the place `distance`.

VII. Architecture Modeling

The way how to model system elements influences the system architecture. Basic architecture is based on pure object oriented approach consisting of classes and relationships between classes. DEVS architecture is based on components that are connected using the formalism of DEVS.

A. Basic Architecture Modeling

Figure 18 shows the classes of basic architecture of our example. Classes from different levels are identified with appropriate stereotypes—*Activity Net*, *Role*, and *Subject*. Each class can be modeled in different formalism, therefore the stereotypes of model formalism are introduced too. In this example, classes are modeled only using Petri nets, so that the stereotype *PN* is used.

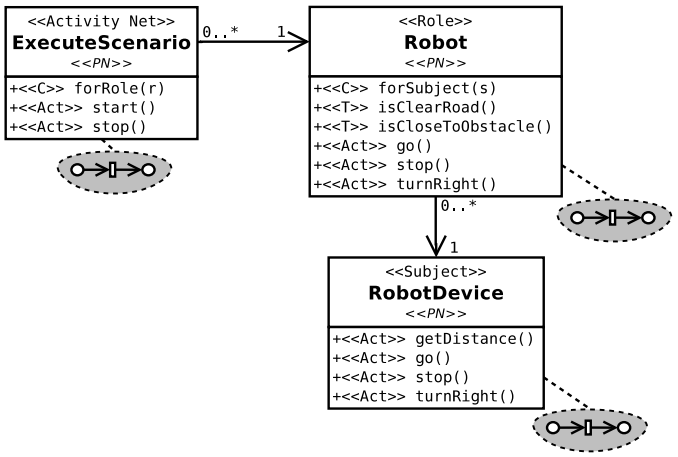


Figure 18. Basic architecture of the case study.

The example architecture consists of the subject *RobotDevice*, its role *Robot* and its activity *Scenario*, that have

been modeled by OOPN (see the stereotype *PN*). *RobotDevice* represents an interface to the simulated robot and *Robot* represents a role, which the robot has in the system. Each method is labeled with one of stereotypes `C` (constructor), `Act` (activity), and `T` (testing) determining a realization of methods (it was introduced in [18]).

B. Combination of Formalisms of DEVS and OOPN

Discrete Event System Specification (DEVS) [19] is a formalism, which can represent any system whose input/output behavior can be described as sequence of events. The atomic DEVS model is specified as a structure M containing sets of states S, input and output event values X and Y, internal transition function δ_{int}, external transition function δ_{ext}, output function λ, and time advance function ta. These functions describe behavior of the component.

This way we can describe atomic models. Atomic models can be coupled together to form a coupled model CM. The later model can itself be employed as a component of a larger model. This way the DEVS formalism brings a hierarchical component architecture. Sets S, X, Y are obviously specified as structured sets. It allows to use multiple variables for specification of a state; we can use a concept of input and output ports for input and output events specification, as well as for coupling specification. Let us have the structured set $X = (V_X, X_1 \times \cdots \times X_n)$, where V_X is an ordered set of n variables and $X_1 \times \cdots \times X_n$ denotes a value for each member from the set V_X. We can write the structured set as $X = \{(v_1^x, \ldots, v_n^x) | v_1^x \in X_1, \ldots, v_n^x \in X_n)\}$. Members $v_1^x, \ldots, v_n^x$ are called *input ports*, resp. *output ports* for the set of output events Y. $V_X(v_1^x)$, resp. $V_Y(v_1^y)$, then denotes a value of the input port v_1^x, resp. output port v_1^y. In another words, components are connected by means of ports and event values are carried via these ports.

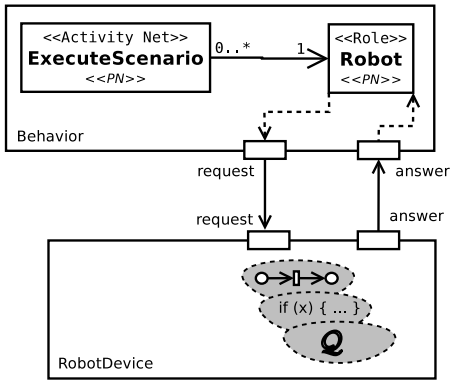


Figure 19. DEVS architecture of the case study – the model *DEVSRobot*.

DEVS components can be described by any formalisms with respecting DEVS functions. Thus, DEVS component can wrap another kind of formalism, so that each such a formalism is interpreted by its simulator and simulators communicate each other by means of a compatible interface. Let $M_{PN} = (M, \Pi, map_{inp}, map_{out})$ be a DEVS component M, which wraps an OOPN model Π. The model Π defines an initial class c_0, which is instantiated immediately the component M_{PN} is created. Functions map_{inp} and map_{out} map ports and places of the object net of the initial class c_0. The mapped places then serve as input or output ports of the component.

C. DEVS Architecture Modeling

The model can be split up into components in accordance to their responsibility. For instance, the system of our case study has two basic parts (i.e., components)—the model of behavior (activity nets and roles) and the model of real robot (subjects; the subject can provide a communication channel to a real robot or can simulate it). Components can be modeled using means of UML, i.e., packages. In that case, the component interface is provided by classes themselves, so that the replacement of components is complicated. Formalisms such as DEVS define a stable interface allowing to exchange components in a very simple way, because components are connected only by means of ports. It is one of reasons we have made a decision to use DEVS formalism to describe the system architecture.

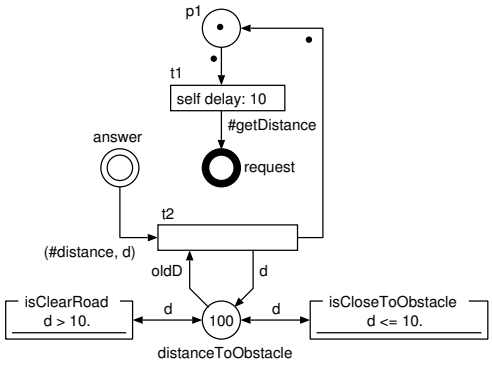


Figure 20. The role *Robot* – implementation for DEVS architecture.

The DEVS architecture of presented case study contains two components *Behavior* and *RobotDevice* as shown in Figure 19. The component *Behavior* describes the system behavior, as presented in a case of basic architecture. The component *RobotDevice* describes the robot subject and can be modeled by OOPN, programming language, or any other supported formalism. Components are connected via ports *request* and *answer*. Both components are coupled the model called *DEVSRobot*.

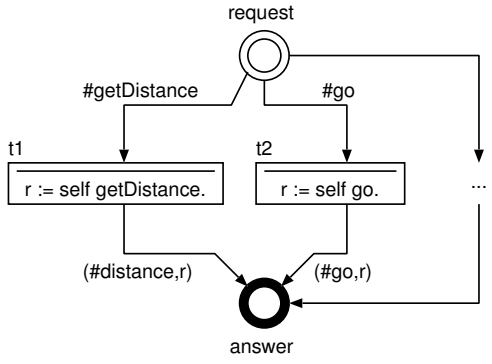


Figure 21. Extension of the subject model *RobotDevice* for the DEVS architecture reason.

Because the architecture changes, we have to modify classes describing system behavior in the component *Behavior*. This component encapsulates OOPN model and the initial class has to be defined. Because the interface between components serves for communication to the subject of robot, and subjects can communicate to roles, the *Robot* class will be the initial one. It means, that ports *request* and *answer* are mapped to places of the *Robot* object net. This modified object net is shown in Figure 20. Place named *request*, resp. *answer*, corresponds to output port *request*, resp. input port *answer*.

$$
\begin{aligned}
variables: \quad & r \leftarrow nil \\
& d \leftarrow 20 \\
& dists[] \leftarrow array\ of\ (100, 40, 0, 40, 200, 0) \\
& i \leftarrow 0 \\
& go \leftarrow false \\
getDist(): \quad & return\ (\#distance, d) \\
turnR(): \quad & i < 6 : i \leftarrow i + 1 \\
& d \leftarrow dists[i] \\
go(): \quad & go \leftarrow true \\
stop(): \quad & go \leftarrow false
\end{aligned}
$$

Figure 22. Internal data and functions of the component *RobotDevice*.

The component *RobotDevice* can be described by OOPN. The posible model results from the class *RobotDevice* shown in Figures 15 and 17. This class is an initial class of the component and adds model elements to the object net as shown in Figure 21. It gets a request string from its input port $request$, asks itself for answer, and puts the answer to its output port $answer$.

Another way is to describe the component *RobotDevices* as an atomic DEVS. First, the internal data and functions are defined as shown in Figure 22. They correspond to the methods and data stored in places of basic model of the subject *RobotDevice* shown in Figures 15 and 17.

DEVS functions are defined as shown in Figure 23. They use internal data and functions. If an external event occurs, the function δ_{ext} is performed; it puts a value from input port *request* to the variable r. Time advance function ta is defined as follows: if there is a request ($r \neq nil$), then internal and output functions will be called immediatelly ($ta \leftarrow 0$); if there is no request and the variable go is $true$, then internal and output functions will be called in 1 time unit ($ta \leftarrow 1$); if there is no request and the variable go is $false$, then nothing happen ($ta \leftarrow \infty$). Output function λ calls internal functions depending on the request r. If called functions return any value a, this value a is put to the output port *answer*. If the internal transition function δ_{int} is called and there is no request, it decrease the distance to an obstacle ($d \leftarrow d - 2$). In any case, it destroys an information about processed request ($r \leftarrow nil$).

VIII. Software Deployment with Models

This section will demonstrate possibilities of keeping models in the deployed system. The goal is to use the same formalism for system modeling as well as for system implementation and deployment. It is based on the application framework allowing to interoperability of models and product environment.

$$
\begin{aligned}
\delta_{ext}: \quad & r \leftarrow V_X(request) \\
\lambda: \quad & r = \#getDistance : a \leftarrow getDist() \\
& r = \#turnRight : a \leftarrow turnR() \\
& r = \#go : a \leftarrow go() \\
& r = \#stop : a \leftarrow stop() \\
& a \neq nil : V_Y(answer) \leftarrow a \\
\delta_{int}: \quad & r = nil\ \&\ d > 2 : d \leftarrow d - 2 \\
& r \leftarrow nil \\
ta: \quad & \begin{cases} 0, & r \neq nil \\ 1, & go\ \&\ r = nil \\ \infty, & not\ go\ \&\ r = nil \end{cases}
\end{aligned}
$$

Figure 23. DEVS functions of the component *RobotDevice*.

A. Application Framework

The application framework has to fulfil two basic requirements. First, to link models and product environment. Second, to work with models in simulations.

First, the models described by means of OOPN can cooperate with objects of the product environment (product objects). Since the developed framework [20] is implemented in Smalltalk [21], OOPN objects can send messages to Smalltalk objects, and OOPN objects can be directly available in Smalltalk. There are different levels at which the product objects can send messages to OOPN objects—*domain*, *predicate*, and *synchronous port* levels. Domain level allows Smalltalk objects to send messages OOPN objects as though they were Smalltalk objects. Predicate level allows to test predicates and port level allows to perform synchronous ports. Each OOPN object offers special meta-protocol allowing to work at presented levels (it will be shown in the text, later on).

Second, the framework allows to execute models in different simulation modes—simulation in *model time*, simulation in *real time*, and simulation in *combined time*. Each simulation mode is advisable for another kind of usage. *Model time* is intended for basic design, testing, and analysis of system under development and assumes all components are described by formal models. *Combined time* assumes that the system is descibed by formal models as well as implemented in product environment, i.e., selected simulated components are replaced by their real implementation, whereas simulated components work in model time and real components work in real time. This mode allows to experiment with simulation models in real conditions. *Real time* assumes that all components (simulated as well as real) work in real time and is intended for hardware/software-in-the-loop simulation and system deployment.

B. Implementation with Basic Architecture

We can exchange the simulated subject by an interface to the real robot. It is very simple—we only create instances of appropriate classes and do not care about used formalism. Figure 24 of Smalltalk code shows creating a subject as an instance of a Smalltalk class. This subject cooperates with a role and an activity modeled by OOPN. The object *Repos* represents the storage of all classes and simulations using OOPN or DEVS formalisms.

```
cAct  := Repos componentNamed:
                    'ExecuteScenario'.
cRole := Repos componentNamed: 'Robot'.
subj := RobotDevice new.
role := cRole forSubject: subjR.
actS := role createActivity.
actS start.
```

Figure 24. Accessing OOPN objects from Smalltalk.

Now, we demonstrate an accessing OOPN objects from product environment of Smalltalk. We send a command `go` to start walking—the message passing is provided in the standard form. To test an object state, the predicates should be used. Since they are not ordinary methods, we have to access them in a special way. We obtain a special meta-protocol by sending a message `asPredicate` and then call synchronous port or negative predicate in the standard form of message passing. The result represents a state of a called port/predicate, which has been tested. In our example, we test the predicate `isCloseToObstacle` and if the result is true, then we stop robot's walking by sending a message `stop`. The example is shown in Figure 25.

```
role go.
r := role asPredicate isCloseToObstacle.
r ifTrue: [ role stop ].
```

Figure 25. Message passing and predicate testing.

Of course, proposed solution is not sufficient for our case, because we need to test this condition until it becomes true. Therefore, we can use one of following ways—to use waiting for specified condition or to define *a listener*. The first way is shown in Figure 26. We simply use a message `waitFor:` from the meta-protocol, which blocks until the specified condition becomes true, i.e., the port `isCloseToObstacle` becomes fireable.

```
role go.
role asPredicate waitFor: #isCloseToObstacle.
role stop.
```

Figure 26. Waiting for a condition.

Second way is shown in Figure 27. It uses a message `listener:for:` from meta-protocol to define a listener, which is activated if the condition becomes true, i.e., the port becomes fireable.

```
role go.
role asPredicate
      listener: self
      for: #isCloseToObstacle.
```

Figure 27. Setting a listener.

The activation of listener means that the special message `conditionSatisfied:` is sent to object, which is specified as a first argument. The example of its implementation is shown in Figure 28.

```
method conditionSatisfied: aCond
    (aCond == #isCloseToObstacle)
        ifTrue: [ role stop ].
```

Figure 28. Listener implementation.

C. Implementation with DEVS Architecture

The example of accessing DEVS components and their object interface is shown in Figure 29. First, we get the DEVS model named *DEVSRobot*, which is based on architecture from Figure 19. Second, we obtain DEVS component *Behavior*, which is able to communicate through its ports. Since this component is described by OOPN, it is possible to use object interface of its initial object (an instance of the class *Robot*) too. To get the object interface, we send a special message `objectInterface` from the component meta-object protocol.

```
s1 := Repos componentNamed: 'DEVSRobot'.
cB := c1 componentNamed: 'Behavior'.
role := cB objectInterface.
```

Figure 29. Obtaining object interface to the inital object.

The variable `role` referees an instance of initial class *Robot* of the component *Behavior*. The other manipulation is the same as in the case of the basic architecture. For instance, to wait for the condition *isCloseToObstacle* a sequence of messages shown in Figure 26 can be used. It blocks until the port `isCloseToObstacle` becomes fireable and then stops the robot.

IX. Conclusion and Future Work

The paper dealt with an approach to system development and deployment using formal models as a basic design, analysis and programming means combining simulated and real components. Combination of two formalisms has been taken into account—Object Oriented Petri Nets (OOPN) for behavior description and Discrete Event System Specification (DEVS), which can be used for architecture description as well as behavior description. The main advantage of that approach is no need for code generation and further investigation of deployed systems using the same formal models. The process of such an development was demonstrated on the case study of simple robotic system.

The proposed approach has one main disadvantage—usage of application framework, which interprets formal models directly demands of increased requirements on memory size and system performance. The future research will aim at efficient representation of choosed formal models and interoperability with another product environment. The application framework will be adapted to new conditions having lesser requirement for resources.

ACKNOWLEDGMENT

This work has been supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), by BUT FIT grant FIT-S-11-1, and by the Ministry of Education, Youth and Sports under the contract MSM 0021630528.

REFERENCES

[1] R. Kočí and V. Janoušek, "Object oriented Petri nets in software development and deployment," in ICSEA 2013, The Eighth International Conference on Software Engineering Advances. Xpert Publishing Services, 2013, pp. 485–490.

[2] S. Beydeda, M. Book, and V. Gruhn, Model-Driven Software Development. Springer-Verlag, 2005.

[3] M. Broy, J. Gruenbauer, D. Harel, and T. Hoare, Eds., Engineering Theories of Software Intensive Systems: Proceedings of the NATO Advanced Study Institute. Kluwer Academic Publishers, 2005.

[4] C. Raistrick, P. Francis, J. Wright, C. Carter, and I. Wilkie, Model Driven Architecture with Executable UML. Cambridge University Press, 2004.

[5] D. S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing, ser. 17 (MS-17). John Wiley & Sons, 2003.

[6] R. Kočí and V. Janoušek, "System design with Object oriented Petri nets formalism," in The Third International Conference on Software Engineering Advances Proceedings ICSEA 2008. IEEE Computer Society, 2008, pp. 421–426.

[7] R. Kočí and V. Janoušek, "OOPN and DEVS formalisms for system specification and analysis," in The Fifth International Conference on Software Engineering Advances. IEEE Computer Society, 2010, pp. 305–310.

[8] M. Češka, V. Janoušek, and T. Vojnar, PNtalk — a computerized tool for Object oriented Petri nets modelling, ser. Lecture Notes in Computer Science. Springer Verlag, 1997, vol. 1333, pp. 591–610.

[9] R. Kočí and V. Janoušek, Simulation Based Design of Control Systems Using DEVS and Petri Nets, ser. Lecture Notes in Computer Science. Springer Verlag, 2009, vol. 5717, pp. 849–856.

[10] J. L. Fernandez, R. Sanz, E. Paz, and C. Alonso, "Using hierarchical binary Petri nets to build robust mobile robot applications: RoboGraph," in IEEE International Conference on Robotics and Automation, 2008, pp. 1372–1377.

[11] C. Rust, F. Stappert, and R. Kunnemeyer, "From Timed Petri nets to interrupt-driven embedded control software," in International Conference on Computer, Communication and Control Technologies (CCCT 2003), 2003.

[12] O. Bayo-Puxan, J. Rafecas-Sabate, O. Gomis-Bellmunt, and J. Bergas-Jane, "A GRAFCET-compiler methodology for C-programmed microcontrollers, In Assembly Automation," Assembly Automation, vol. 28, no. 1, 2008, pp. 55–60.

[13] R. Valk, "Petri nets as token objects: an introduction to Elementary object nets." in Jorg Desel, Manuel Silva (eds.): Application and Theory of Petri Nets; Lecture Notes in Computer Science, vol. 120. Springer-Verlag, 1998.

[14] D. Moldt, "OOA and Petri nets for system specification," in Object-Oriented Programming and Models of Concurrency. Italy, 1995.

[15] L. Cabac, M. Duvigneau, D. Moldt, and H. Rölke, "Modeling dynamic architectures using nets-within-nets," in Applications and Theory of Petri Nets 2005. 26th International Conference, ICATPN 2005, Miami, USA, 2005, pp. 148–167.

[16] V. Janoušek and R. Kočí, "PNtalk: concurrent language with MOP," in Proceedings of the CS&P'2003 Workshop. Warsaw University, Warsawa, PL, 2003.

[17] R. Kočí and V. Janoušek, "Modeling and simulation-based design using Object-oriented Petri nets: a case study," in Proceeding of the International Workshop on Petri Nets and Software Engineering 2012, vol. 851. CEUR, 2012, pp. 253–266.

[18] R. Kočí and V. Janoušek, "Specification of UML classes by Object oriented Petri nets," in ICSEA 2012, The Seventh International Conference on Software Engineering Advances. Xpert Publishing Services, 2012, pp. 361–366.

[19] B. Zeigler, T. Kim, and H. Praehofer, Theory of Modeling and Simulation. Academic Press, Inc., London, 2000.

[20] R. Kočí, "PNtalk system," http://perchta.fit.vutbr.cz/pntalk2k, 2004. [Online]. Available: http://perchta.fit.vutbr.cz/pntalk2k

[21] A. GoldBerk and D. Robson, Smalltalk 80: The Language. Addison-Wesley, 1989.

Localizing Software Bugs using the Edit Distance of Call Traces

Themistoklis Diamantopoulos and Andreas Symeonidis
Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki
Information Technologies Institute, Centre for Research and Technology Hellas
Thessaloniki, Greece
Email: {thdiaman,asymeon}@issel.ee.auth.gr

***Abstract*—Automating the localization of software bugs that do not lead to crashes is a difficult task that has drawn the attention of several researchers. Several popular methods follow the same approach; function call traces are collected and represented as graphs, which are subsequently mined using subgraph mining algorithms in order to provide a ranking of potentially buggy functions-nodes. Recent work has indicated that the scalability of state-of-the-art methods can be improved by reducing the graph dataset using tree edit distance algorithms. The call traces that are closer to each other, but belong to different sets, are the ones that are most significant in localizing bugs. In this work, we further explore the task of selecting the most significant traces, by proposing different call trace selection techniques, based on the Stable Marriage problem, and testing their effectiveness against current solutions. Upon evaluating our methods on a real-world dataset, we prove that our methodology is scalable and effective enough to be applied on dynamic bug detection scenarios.**

Keywords-automated debugging, dynamic bug detection, frequent subgraph mining, tree edit distance, Stable Marriage problem.

I. INTRODUCTION

During the latest few decades, software reliability has grown to be a major concern for both academia and the industry. Software bugs can lead to faulty software and dissatisfied customers, since testing and debugging are quite costly even compared to the software development phase. As software grows more and more complex, though, identifying and eliminating software bugs has become a challenging task.

There are two types of software bugs: *crashing* bugs and *non-crashing* bugs. The former, as their name implies, lead to program crashes, thus they are easier to locate by tracing the call stack at the time of the crash. The latter, however, are logic errors that do not lead to crashes. The problem of locating non-crashing bugs is quite difficult, since no stack trace of the failure is available. Thus, finding a bug would usually involve careful examination of the source code, thorough testing, even pure intuition as to where the bug might reside. An interesting line of research aims towards automating the bug locating procedure by applying *Data Mining (DM)* techniques on call traces of *correct* and *incorrect* program executions. Hence, since dynamic analysis is performed to detect such bugs, the field is known as *dynamic bug detection*.

As noted in [1], dynamic bug detection techniques may be broadly classified according to the granularity of the source code instrumentation approach. Highly granular approaches involve inserting checks in different source code positions, either in the form of counters or boolean predicates. Indicatively, Liblit et al. [2] heuristically eliminate counters and apply logistic regression (or statistics as in [3]) to identify the attributes that affect the class (bug vs no bug). On the other hand, Liu et al. [4] employ boolean predicate statistics on correct and incorrect executions to localize bugs.

A more coarse-grained approach concerns inserting checks at block level, where blocks are fragments of code between branches. Renieris and Reiss [5] follow this approach and perform nearest neighbor queries to identify incorrect execution traces that are close to correct ones and compare these couples to detect potentially buggy blocks of code.

Counter-level and block-level approaches are quite precise in localizing bugs. However, the rise of *Object Oriented* and *Functional Programming* has led to preference for small comprehensive functions, indicating that instrumenting functions can also be effective. Mining traces at function level, and thus identifying potentially buggy functions is generally fine-grained enough for localizing a bug, as long as proper programming paradigms are employed. Approaches in this category employ *Graph Mining* techniques to call traces to identify which subgraphs are more frequent in incorrect than in correct runs [6]–[8]. Current approaches include also efforts towards improving the Graph Mining procedure [9], or using different representations such as N-grams (subsequences of length N) [10], or even reformulating the problem as a search/optimization problem [11]. Although these approaches are effective, their scalability is arguable.

The procedure of bug localization is similar for all approaches. The generated call traces constitute a dataset that has to be mined in order to detect bugs; and this is where the problems start. Even at function-level, datasets are usually huge. For a small application, with, e.g., 150 functions, there may be couples of thousands of transitions among them. In this context, creating an effective, yet also scalable, solution is a challenging problem. And, though it has been broadly studied, most literature approaches focus on reducing the size of each trace, without reducing the number of traces in the dataset.

Previous work [1] on reducing the size of the dataset has indicated considerable improvement on both scalability and effectiveness. We extend this work by reformulating the problem and providing different methodologies that focus on achieving effectiveness without compromising scalability. As in [1], the methodology involves performing Graph Mining

techniques to a subset of the dataset, thus the main focus lies on determining that subset. The similarity between any two call traces of the dataset can be defined as their edit distance, i.e., the cost of turning the one trace to the other. Given the distances between all combinations of traces, the problem is reduced to designing a call trace selector algorithm that successfully determines the most "useful" traces, i.e., the ones that successfully isolate the bug. In this paper, we further explore the effect of the call trace selector algorithm, by proposing two new algorithms, and comparing the different methodologies with respect to scalability and effectiveness. Furthermore, we benchmark our methods against known function-level dynamic bug detection techniques in order to discuss their applicability in real applications and evaluate their effectiveness against the current state-of-the-art.

Section II of the paper reviews current literature on function-level dynamic bug detection, illustrating the general procedure followed to mine the traces and identify the Graph Mining problems. Section III provides insight for reducing the call trace dataset and explores the tasks of comparing call traces and selecting the most "useful" subset of the call trace dataset. The construction of a realistic dataset that illustrates our contribution is explained in Section IV. Finally, our implementation is evaluated in terms of efficiency and effectiveness in Section V, while Section VI concludes the paper and provides insight for further research.

II. Function-Level Dynamic Bug Detection

In this section, we discuss the basic steps followed in function-level bug localization techniques, while denoting the different approaches. The following subsections correspond to the main phases of constructing a graph dataset, reducing the graphs to improve scalability, and applying Graph Mining techniques to provide the final ranking of possibly buggy functions.

A. Graph Dataset Construction

Assuming there is a set of test cases, program functions are instrumented and the test cases are executed to produce a set of *call traces* (or *scenarios*) S. A call trace is initially a rooted ordered tree, with the `main` function as its root. Two more sets, $S_{correct}$ and $S_{incorrect}$ are defined, corresponding to correct and incorrect program executions. The correctness of an execution can be determined by the developer. In generated datasets such as the one used in this paper, one can also determine it automatically by comparing the output of the erroneous versions of the program with the correct version. Thus, upon collecting the execution traces, the graph dataset[1] is constructed.

B. Graph Reduction

At this point, the graphs of the dataset are quite large, with thousands of nodes and respective edges. Applying a Graph Mining algorithm to such a dataset would be highly inefficient. Thus, *graph reduction* is performed on each graph in order to keep only useful information while discarding all redundant data, to reduce its size. Figure 1 depicts several state-of-the-art graph reduction techniques.

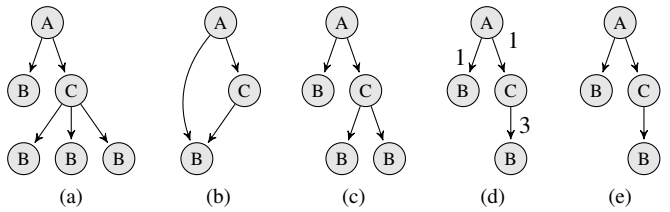


Figure 1. An example call graph (a) and four different reduced graphs with respect to the reduction techniques, including (b) total reduction, (c) one-two-many reduction, (d) subtree reduction and (e) simple tree reduction.

The first technique, known as *total reduction*, is presented by Liu et al. [6]. The authors create a graph using each edge of the initial call graph once and discard any structural information (i.e., tree levels). Total reduction, shown in Figure 1b, is the most efficient reduction method since it actually preserves minimum information.

However, total reduction fails to capture the structure of the call graph, thus different alternatives have been applied to preserve more information, while keeping the graph as small as possible. A straightforward solution is the one proposed by Di Fatta et al. [7]; the authors perform *one-two-many* reduction, preserving tree structure by keeping exactly two child nodes whenever the children of a node are more than two (see Figure 1c).

Eichinger et al. [8] claim that total reduction and one-two-many reduction are not sufficient, since they discard call frequency information. According to the authors, the number of times (i.e., frequency) that a function calls another function is crucial since it can capture bugs that may occur in, e.g., the third or fourth time the function is called. Thus, they propose *subtree reduction*, a technique that preserves both the structure of the tree and the frequency of function calls (see Figure 1d).

As one might observe, the reduction techniques are based on a compromise between information loss and scalability. Although subtree reduction maintains most information, it is quite inefficient since it immediately adds a weight parameter to the graph. Since the scope of this work lies in scalability, we decided to use a reduction technique called *simple tree reduction*, shown in Figure 1e, which was originally introduced in [1]. Reducing a graph using simple tree reduction involves traversing all the nodes once and deleting any duplicates as long as they are on the same level. The reduced graph is a satisfactory representation of the original one since large part of its structure is preserved.

C. Graph Mining

Upon reduction, the problem lies in determining the nodes (functions) that are frequent in the incorrect set $S_{incorrect}$ and infrequent in the correct set $S_{correct}$. Intuitively, if a function is called every time the result is incorrect, it is highly possible

[1] Any tree is obviously also a graph. The terms are used interchangeably concerning the class of techniques that may be applied to the dataset.

to have a bug. However, having more than one function with the same frequency is also possible. Thus, the Graph Mining algorithm should find the closed frequent subgraphs, i.e., the subgraphs for which no supergraph has greater support in $S_{incorrect}$.

Finding frequent subgraphs in a graph dataset is a well-known problem, defined as *Frequent Subgraph Mining (FSM)*. State-of-the-art algorithms include gSpan [12] and Gaston [13]. Furthermore, since these graphs are actually trees, several *Frequent Subtree Mining (FTM)* algorithms, such as FreeTreeMiner [14], may be used as well. Although those algorithms are applicable to the problem, there is strong preference for *CloseGraph* [15], an algorithm that is highly scalable since it prunes unnecessary input and outputs only closed frequent subgraphs.

D. Ranking

The output of the CloseGraph algorithm is a set of frequent subgraphs, along with their support in the correct and the incorrect set. Hence, the question is how can a ranking of possibly buggy functions be created by such a set. It is typical to use DM techniques based on *support* and *confidence* to determine which subgraphs are actually interesting. For instance, Di Fatta et al. [7] suggest ranking the functions according to their support in the failing set. According to Eichinger et al. [8], this type of ranking can be called *structural*. The structural ranking for each function f is defined as:

$$P_s(f) = support(f, S_{incorrect}) \tag{1}$$

The support of each function in the failing set $S_{incorrect}$ provides a fairly effective ranking. However, the scoring defined in equation (1) is not sufficient, since it does not take confidence into account. Furthermore, finding the support only on incorrect executions yields skewed results, since a function with large support in both $S_{correct}$ and $S_{incorrect}$ would be ranked high, even though it may be insignificant with respect to the bug.

Several variations of the structural ranking have emerged in order to overcome the aforementioned issues [4][7]. In this paper, we use an entropy-based ranking technique proposed by Eichinger et al. [8] since it is proven to outperform the other techniques. The main intuition behind this ranking technique is to identify the edges that are most significant to discriminate between correct and incorrect call traces. A table is created with columns corresponding to subgraph edges and rows corresponding to graphs. The table holds the support of each edge in every graph. Consider the example of Table I.

TABLE I. ENTROPY-BASED RANKING EXAMPLE

$Graph$	$f_1 \rightarrow f_2$	$f_1 \rightarrow f_3$	$f_2 \rightarrow f_4$	...	$Class$
G_1	4	7	2	...	$correct$
G_2	9	5	8	...	$incorrect$
G_3	6	3	1	...	$correct$
...	...	...	...	...	...

In Table I, $F = f_1, f_2, \ldots$ is the set of functions and $G = G_1, G_2, \ldots$ is the set of graphs. The table is constructed given the support of each subgraph in the graphs. Thus, supposing subgraph SG_1 appears 4 times in graph G_1 and edge $f_1 \rightarrow f_2 \in SG_1$, the support of the edge in graph G_1 is 4. If, e.g, both SG_2 and SG_3 contain the edge $f_1 \rightarrow f_2$ and they appear 5 and 4 times respectively in graph G_2, then the support of this edge in graph G_1 is 9. As one might observe, the problem is actually a *feature selection* problem, i.e., defining the features (edges) that discriminate between the values of the class feature ($correct$, $incorrect$). Thus, any feature selection algorithm may be used to determine the most significant features. Eichinger et al. [8] calculate the information gain for each feature, and interpret the result for each feature (ranging from 0 to 1) as the probability of it being responsible for a bug. The respective probability $P_e(f)$ for a node (function) is determined by the maximum probability of all the edges it is connected to.

Finally, one can calculate the combined ranking for a function f by averaging over its structural ranking $P_s(f)$ and its entropy-based ranking $P_e(f)$. However, since $P_s(f)$ and $P_e(f)$ may have different maximum values, it is necessary that their values are normalized by dividing each ranking by its maximum value. Thus, the combined ranking $P(f)$ is calculated as follows:

$$P(f) = \frac{1}{2} \cdot \left(\frac{P_e(f)}{\max\limits_{f \in F} P_e(f)} + \frac{P_s(f)}{\max\limits_{f \in F} P_s(f)} \right) \tag{2}$$

where the maximum values at the denominators are used in order to normalize the weighting of each ranking. Finally,$P(f)$ provides the probability that a function f contains a bug in the range $[0, 1]$.

III. REDUCING THE GRAPH DATASET

The steps defined in Section II are common for all function-level bug detection algorithms. Several researchers have indicated the need for scalable solutions, which is generally accomplished by reducing the graphs (see Subsection II-B). Ideally, graph reduction captures the most important information of the graph while minimizing its size. However, even upon reduction, the number of graphs in the dataset is quite large, thus making the mining step quite inefficient.

When a dataset of several graphs is given, not all of them are equally useful in locating the bug. Consider a simple scenario for the `grep` program. Assume the program has a bug that results in faulty executions when the `?` character is used in a *Regular Expression (RE)*, such that the appropriate words are not returned, if the preceding element appears 0 times. Normally, if a symbol is succeeded by the `?` character, then it may be found 0 or 1 times exactly. Consider running the `grep` program for one word at a time for the following phrase:

```
there once was a cat that ate a rat
  and then it sat on a yellow mat
```

In this text, the RE `[a-z]*c?at` should match the words in the set $S_{matched} = \{$`cat`, `that`, `rat`, `sat`, `mat`$\}$, i.e., all words having any letter from `a` to `z` 0 or more times, followed by the letter `c` 0 or 1 times exactly, and followed by letters `a` and `t`. Instead it only matches the word `cat`. Consider also

the set of words that are not matched $S_{unmatched} = \{$there, once, was, a$_{(1)}$, ate, a$_{(2)}$, and, then, it, on, a$_{(3)}$, yellow$\}$. Assuming that all the possible traces are created, several of them, such as the ones created from the $S_{matched}$ set, are actually much more significant in identifying the bug, since it actually resides only on the $S_{matched}$ set. Thus, traces of cat and rat should be more *similar* than traces of cat and yellow. In fact, when executing the cat and rat scenarios, many function calls coincide. However, this is also true for the traces of was and it. Intuitively, determining which traces are highly indicative of the bug can be based on the similarity between them as well as whether they are correct or incorrect. Thus, correct executions that are *similar* to the incorrect ones (e.g., rat may be close to cat) should isolate more easily the buggy functions. On the other hand, when two correct (or incorrect) executions are quite close to each other (e.g., the traces from was and it could be quite similar), then one of them should provide all necessary information.

The above example is formed such that it is easy to understand. One could ask why not select test cases by hand, so that they are discriminating. However, this is usually impossible since real scenarios are much more complex, e.g., for the grep case there may be passages instead of words. In addition, certain executions may seem similar, yet be significantly different with respect to the call traces. Thus, at first, there is the need for a similarity metric between two traces. Having such a metric, one can apply an algorithm to select the most discriminating call traces based on the aforementioned intuitive remarks. Similarly to [1], the metric used to compare the similarity between two trees is the edit distance between them. Subsection III-A provides different alternatives for computing this metric, while Subsection III-B illustrates our proposed algorithms for using it to reduce the size of the dataset.

A. The Tree Edit Distance Problem

A metric widely used to represent the similarity between two strings is the *String Edit Distance (SED)* between them. SED is defined as the number of edit operations required to transform one string to the other. SED operations usually contain insertion or deletion of characters. Concerning trees, such as the ones of our dataset, *Tree Edit Distance (TED)* algorithms can be used to calculate the distance between two of them. The following paragraphs provide a definition of the TED problem as well as two well known algorithms of current literature in finding TED.

The TED problem was originally posed by Tai [16] in 1979. The possible *edit operations* are defined in Figure 2.

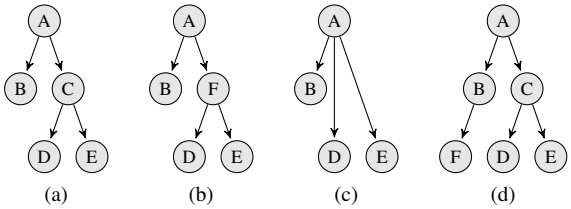


Figure 2. An example tree (a) and three different edit operations: (b) node relabeling, (c) node deletion, and (d) node insertion.

The first operation, *node relabeling*, concerns simply changing the label of a node (see Figure 2b). *Node deletion* is performed by deleting a node of the tree and reassigning any children it had so that they become children of the deleted node's parent. For example, in Figure 2c, the children of deleted node C are reassigned to C's parent A. Finally, *node insertion* concerns inserting a new node in a position in the tree, such as inserting node F in Figure 2d. Assuming a *cost function* is defined for each edit operation, an *edit script* between two trees T_1, T_2 is a sequence of operations required to turn T_1 into T_2, and its cost is the aggregate cost of these operations. Thus, the TED problem is defined as determining the *optimal edit script*, i.e., the one with the minimum cost.

1) Zhang-Shasha Algorithm: One of the most well known TED algorithms is the *Zhang-Shasha* algorithm, which was named after its authors, K. Zhang and D. Shasha [17]. Let $\delta(T_1, T_2)$ be the edit distance between trees T_1 and T_2, and $\gamma(l_1 \rightarrow l_2)$ be the cost of the edit operation from l_1 to l_2. A simple recursive algorithm for computing TED is defined using the following equations:

$$\delta(\theta, \theta) = 0 \tag{3}$$

$$\delta(T_1, \theta) = \delta(T_1 - u, \theta) + \gamma(u \rightarrow \lambda) \tag{4}$$

$$\delta(\theta, T_2) = \delta(\theta, T_2 - v) + \gamma(\lambda \rightarrow v) \tag{5}$$

$$\delta(T_1, T_2) = min \begin{cases} \delta(T_1 - u, T_2) + \gamma(u \rightarrow \lambda) \\ \delta(T_1, T_2 - v) + \gamma(\lambda \rightarrow v) \\ \delta(T_1(u), T_2(v)) + \delta(T_1 - T_1(u), \\ \qquad T_2 - T_2(v)) + \gamma(\lambda \rightarrow v) \end{cases} \tag{6}$$

where $T - u$ denotes tree T without node u and $T - T(u)$ denotes tree T without u or any of each children. Parameter λ is the performed edit operation. The *Zhang-Shasha* algorithm uses *Dynamic Programming (DP)* in order to compute the TED. The *keyroots* of a tree T are defined as:

$$keyroots(T) = \{root(T)\} \cup \{u \in T : \text{u has left siblings}\} \tag{7}$$

Given (7), the *relevant subtrees* of T are defined as:

$$relevant\_subtrees(T) = \bigcup_u \{T(u)\}, \forall u \in keyroots(T) \tag{8}$$

Thus, the algorithm recursively computes the TED by finding the relevant subtrees and applying equations (3)–(6).

2) pq-Grams Algorithm: Several algorithms solve the TED problem effectively. However, even the most efficient ones lack scalability, since the polynomial order of the problem is high. A promising way of reducing the complexity of the problem and improving efficiency is by approximating the TED instead of computing its exact value. Approximate TED algorithms can generally be effective enough when results do not need to be exact. In the call trace scenario, the TED is a value denoting the similarity of two trees, thus, even if it is approximate, it shall be sufficient for the call trace selector algorithms of Subsection III-B.

Such an approximate TED algorithm is the *pq-Grams* based algorithm proposed by Augsten et al. [18]. The authors define pq-Grams as a port of known string q-grams to trees. An example tree and its pq-Grams are shown in Figure 3.

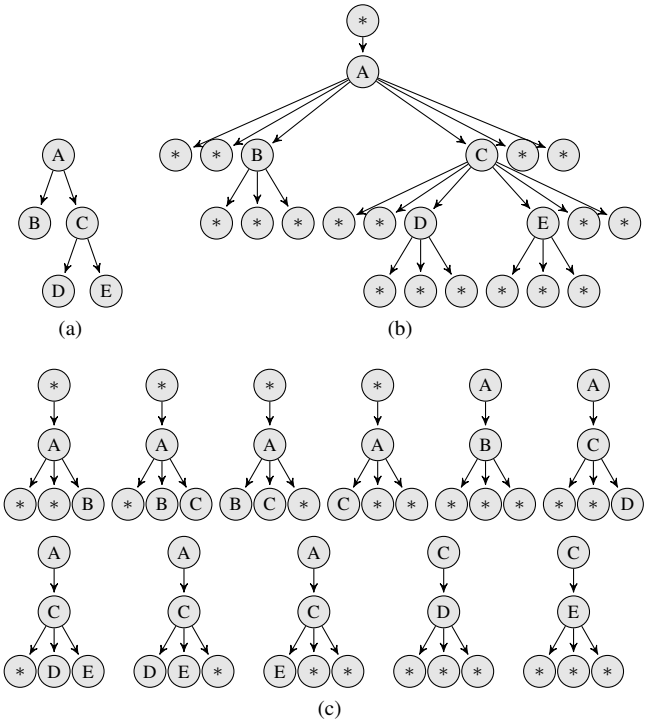


Figure 3. A pq-Grams example for $p = 2$ and $q = 3$, containing (a) an example tree, (b) its extended form for $p = 2$ and $q = 3$, and (c) its pq-Grams.

Parameters p and q define the *stem* and the *base* of the pq-Gram, respectively. Let $p = 2$ and $q = 3$, the stem of the first pq-Gram of Figure 3c is $\{*, \texttt{A}\}$ and its base is $\{*, *, \texttt{B}\}$. Since the pq-Grams for the tree of Figure 3a cannot be directly created, an intermediate step of extending the tree with dummy nodes is shown in Figure 3b. Finally, The *pq-Gram profile* is the set of all pq-Grams of a tree (see Figure 3c), while the *pq-Gram index* of the tree is defined as the bag of all label tuples for the tree. For example, the pq-Gram index for the tree of Figure 3 is defined as:

$$I(T) = \{*A{*}{*}B, *A{*}BC, *ABC{*}, *AC{*}{*}, AB{*}{*}{*}, AC{*}{*}D, AC{*}DE, ACDE{*}, ACE{*}{*}, CD{*}{*}{*}, CE{*}{*}{*}\} \quad (9)$$

According to Augsten et al. [18], the TED between two trees is effectively approximated by the distance between their pq-Gram indexes. Let $I(T)$ be the pq-Gram index of tree T, the pq-Gram distance between trees T_1 and T_2 is defined as:

$$\delta(T_1, T_2) = |I(T_1) \cup I(T_2)| - 2|I(T_1) \cap I(T_2)| \quad (10)$$

Equation (10) provides a fast way of approximating the TED between any pair of trees of the dataset.

B. The Call Trace Selection Problem

In the previous subsection, we provided two different methods for defining and computing the similarity between two call traces. Assuming that the similarity between all correct-incorrect pairs of the dataset is computed, the problem lies in using this information to determine which call traces can successfully isolate the bug. Formally, assuming that our input consists of the correct and incorrect sets, $S_{correct}$ and $S_{incorrect}$, we must design an algorithm that shall output two new sets $S'_{correct}$ and $S'_{incorrect}$. Let n be the size of each of the new sets, where set $S'_{correct}$ contains the n most important correct graphs and set $S'_{incorrect}$ contains the n most important incorrect graphs. The following paragraphs describe three different call trace selector algorithms that we implemented to solve the problem.

1) The SimpleSelector algorithm: The first algorithm implemented is the *SimpleSelector* algorithm, which was originally described in [1]. The application of the algorithm is shown in Figure 4.

`Input:` n, $S_{correct}$, $S_{incorrect}$
`Output:` $S'_{correct}$, $S'_{incorrect}$
$D = \{(g_c, g_i) \ \forall g_c \in S_{correct}, g_i \in S_{incorrect}\}$
`sort(`D`, key=similarity(`g_c, g_i`))`
$S'_{correct}$ =`First(`$n, \{g_c : \ g_c \in d \in D\}$`)`
$S'_{incorrect}$ =`First(`$n, \{g_i : \ g_i \in d \in D\}$`)`

Figure 4. The SimpleSelector algorithm that sorts the pairs of (correct, incorrect) traces and outputs the first n correct and the first n incorrect traces.

As shown in the figure, the algorithm requires as input the correct and incorrect sets, $S_{correct}$ and $S_{incorrect}$, along with parameter n, which controls how many graphs are going to be retained per set. Initially, the set D, which contains all correct-incorrect pairs of graphs, is sorted according to the similarity of each pair. The set $S'_{correct}$ contains the first n correct unique graphs that are found in the sorted set D, i.e., the n correct graphs that belong to the most similar pairs d of D. The set $S'_{incorrect}$ contains the first n incorrect unique graphs that are found in the sorted set D. For example, given $n = 2$ and $D = \{d_1, d_2, d_3\} = \{(g_1, g_3), (g_1, g_4), (g_2, g_5)\}$ so that the similarity of pair d_1 is larger than that of d_2 and the similarity of d_2 is larger than that of d_3, the sets $S'_{correct}$ and $S'_{incorrect}$ are $\{g_1, g_2\}$ and $\{g_3, g_4\}$ respectively. Function `sort` sorts the set according to the `key`. Finally, function `similarity` can be easily determined using either of the two methods presented in Subsection III-A.

Previous work [1] has indicated that the SimpleSelector algorithm is adequate in terms of effectiveness. Intuitively, since the algorithm selects the most similar pairs of traces, it certainly captures some of the most important traces. However, the algorithm does not account for similar graphs of the same set. In specific, given the above example, graphs g_3 and g_4 could potentially be quite similar to each other. Assuming g_3 and g_4 have identical similarity metrics with g_1, keeping both of them on the final set could produce redundant information. Thus, the second call trace selector algorithm was implemented in order to eliminate this redundancy.

2) The StableMarriage algorithm: The second algorithm implemented is based on the *stable marriage* (or *stable matching*) problem. The problem, first introduced by D. Gale and L. S. Shapley [19] in 1962, has many variants (see [20] for an

extensive survey) since it has numerous applications to real-life problems. According to its original definition, there are N men and N women, and every person ranks the members of the opposite sex in a strict order of preference, i.e., with no equal preference for any member of the opposite sex. The problem is to find a matching between men and women so that there are no two people of opposite sex who would both rather be matched to each other than their current partners. If there are no such people, the matching (marriage) is considered to be *stable*.

The call trace selection problem can be formulated as a stable marriage problem. In this case, the two sexes are the traces of the correct and the traces of the incorrect set. For any trace, given its similarity with the traces of the opposite set, a ranked preference list is formed. Given that the probability two pairs having the same similarity value is too low, we can safely assume that the lists are strictly ordered. Upon forming all the preference lists, the *Gale-Shapley* algorithm [19] can be applied to our problem. The application of the algorithm is shown in Figure 5.

```
Input: n, S_correct, S_incorrect
Output: S'_correct, S'_incorrect
Find preference list ∀g ∈ {S_correct ∪ S_incorrect}
Set all g_c ∈ S_correct and g_i ∈ S_incorrect to unmatched
while ∃ free g_c that has a g_i to try to match
  g_i = g_c's highest ranked incorrect trace
            that g_c has not yet tried to match
  if g_i is unmatched
    (g_c, g_i) are matched together
  else there is a matching (g'_c, g_i)
    if g_i prefers g_c to g'_c
      (g_c, g_i) are matched together
      g'_c becomes unmatched
D = {all pairs (g_c, g_i) of stable matching}
sort(D, key=similarity(g_c, g_i))
S'_correct =First(n, {g_c : g_c ∈ d ∈ D})
S'_incorrect =First(n, {g_i : g_i ∈ d ∈ D})
```

Figure 5. The StableMarriage algorithm that finds the stable matching among the traces of the correct and the incorrect sets, and outputs the traces of the first n (correct, incorrect) pairs.

Functions `sort` and `similarity` work similarly to the SimpleSelector algorithm. The algorithm of Figure 5 initially creates the preference lists for each graph of the sets $S_{correct}$ and $S_{incorrect}$. Note that this step can be reduced to minimum complexity using suitable data structures. At the beginning of the algorithm, all graphs of both sets are unmatched. The algorithm iterates over all correct graphs and tries to match each correct graph g_c to its most preferred incorrect graph g_i for which there was not yet an attempt to match. Any incorrect graph accepts a proposal to match if it is unmatched or if the proposed matching is preferable to its current matching. Given sets $S_{correct} = \{g_1, g_2, g_3\}$ and $S_{incorrect} = \{g_4, g_5, g_6\}$ and the similarity between all correct-incorrect pairs one can construct the ranked preference lists of Figure 6.

$g_1 : g_5 > g_4 > g_6$ $\quad$ $g_4 : g_2 > g_1 > g_3$
$g_2 : g_5 > g_6 > g_4$ $\quad$ $g_5 : g_1 > g_2 > g_3$
$g_3 : g_6 > g_4 > g_5$ $\quad$ $g_6 : g_2 > g_3 > g_1$

(a) (b)

Figure 6. Example preference lists for the graphs of (a) the correct set $S_{correct} = \{g_1, g_2, g_3\}$ and (b) the incorrect set $S_{incorrect} = \{g_4, g_5, g_6\}$.

Iterating over the correct graphs, g_1 initially is matched to g_5 since it is in the top of its preference list. After that, g_2 tries also to match with g_5, and since the g_5 preference list indicates preference for g_2 over g_1, the new pair is (g_2, g_5) and g_1 becomes unmatched. After that, g_3 gets matched with g_6. Since there is still an unmatched graph g_1, and it already tried to match with its first choice it tries to match with its second choice g_4, which is accepted since g_4 is unmatched. Thus, the final pairs that form the set D are sorted according to their similarity, for example resulting in $D = \{(g_1, g_4), (g_3, g_6), (g_2, g_5)\}$. Given, e.g., that the number of retained graphs per set is $n = 2$, the sets $S'_{correct}$ and $S'_{incorrect}$ are $\{g_1, g_3\}$ and $\{g_4, g_6\}$ respectively.

The Gale-Shapley algorithm is proven to converge to a *male-optimal* solution [19], indicating in our case that there is no better matching for the correct graphs of the set. Furthermore, since the final list of the algorithm has no duplicates, any redundant data, i.e., graphs that belong to the same set, are discarded. Although the StableMarriage algorithm seems well adapted to the problem, the algorithm disregards the fact that the pairs are not only ranked but also weighted. Thus, StableMarriage can actually provide a sub-optimal solution to the problem. As a result, we have implemented another algorithm that accounts for the weights and is better suited to the problem at hand.

3) The MaxLinkMax algorithm: The third algorithm that we implemented was initially proposed as a solution to an extension of the stable marriage problem, based on the concept of defining a different form of *stability* for the problem [21]. One can define different notions of stability for a stable matching problem with weighted preferences (see [21] for an extensive definition of several alternatives). In our case, we used the *link-max stability* as a criterion denoting whether the matching is stable. Returning to the marriage metaphor, given a man and a woman, their *link-max strength* is the maximum between the preference value that the man gives to the woman and the preference value that the woman gives to the man. Given a stable marriage problem with weights, a matching is said to be *link-max stable* if there are no two people of the opposite sex of which the marriage would have larger link-max strength than either of the current matchings that each partner has. Formally, let $lm(m, w)$ be the link-max strength of a man m and a woman w, a link-max stable marriage does not contain any pair (m, w) such that:

$$lm(m, w) > lm(m', w) \tag{11}$$

$$lm(m, w) > lm(m, w') \tag{12}$$

where m' is the current partner of w and w' is the current partner of m.

The call trace selection problem can be formulated as a link-max stable matching problem, where the weighted preferences are the similarity values that were determined in Subsection III-A. Thus, one can easily calculate the link-max strength for any pair of (correct, incorrect) graphs, in our case defined as the similarity of the pair. After that, the problem can be solved similarly to the *Max-link-max* algorithm, which was introduced in [21] as a solution to link-max stable marriage problems. The application of the algorithm is shown in Figure 7.

```
Input: n, S_correct, S_incorrect
Output: S'_correct, S'_incorrect
Find preference list ∀g ∈ {S_correct ∪ S_incorrect}
D = ∅
M = {(g_c, g_i) ∀g_c ∈ S_correct, g_i ∈ S_incorrect}
while M ≠ ∅
   (g_c, g_i) = arg max_{(g_c,g_i)∈M} lm((g_c, g_i))
   D = D ∪ (g_c, g_i)
   M = M \ {(g_c, g'_i), (g'_c, g_i) ∀g'_c ∈ S_correct, g'_i ∈ S_incorrect}
S'_correct = First(n, {g_c : g_c ∈ d ∈ D})
S'_incorrect = First(n, {g_i : g_i ∈ d ∈ D})
```

Figure 7. The MaxLinkMax algorithm that finds the link-max stable matching among the traces of the correct and the incorrect sets, and outputs the traces of the first n (correct, incorrect) pairs.

The first step of the algorithm shown in Figure 7 is to create a weighted preference list for each graph in the dataset. After that, two sets are defined: the initial set M, which contains all possible correct-incorrect pairs of graphs and the set D, which is initially empty. The algorithm iterates until the set M is empty. For each iteration, the correct-incorrect pair with the maximum link-max strength is found and added to set D. Let (g_c, g_i) be the selected pair, all the pairs of set M that contain either g_c or g_i are removed from graph M. After all the iterations are over, i.e., the set M is empty, the set D contains the final matching. By contrast with the other algorithms, the pairs are already sorted, thus the final sets $S'_{correct}$ and $S'_{incorrect}$ are immediately defined as the first n correct graphs and the first n incorrect graphs that belong to the first n pairs of D.

The MaxLinkMax algorithm is rather more complex than the other two, since it requires not only a ranking but also the weighted preference for each pair of graphs in the dataset. In line with the remarks of [21], one can find the pair with the maximum link-max strength by saving only the maximum weight for every correct and for every incorrect graph in the dataset. This indicates that the complexity of the algorithm is no higher than that of the two previous algorithms that we implemented. For example, given the graphs of Figure 6, and the weights of the pairs (g_1, g_5), (g_2, g_5), (g_3, g_6), (g_4, g_2), (g_5, g_1) (which is equal to (g_1, g_5)), and (g_6, g_2), one requires to compare only these six values in order to find the pair with the maximum link-max strength. Assuming this pair is the (g_1, g_5), the new pairs are immediately reduced to (g_2, g_6), (g_3, g_6), (g_4, g_2), and (g_6, g_2) (which is equal to (g_2, g_6)). Let $lm(g_2, g_6) > lm(g_3, g_6) > lm(g_4, g_2)$, the pairs are reduced to the minimum (g_3, g_4) and (g_4, g_3) which are equal, directly providing with the final sorted set $D = \{(g_1, g_5), (g_2, g_6), (g_3, g_4)\}$.

In terms of the computational complexity of the algorithms, one could argue that it is quite satisfactory. However, since the expected number of call traces is usually small (e.g., usually less than a hundred), the main scope of these approaches lies in improving effectiveness rather than performance.

IV. Dataset

The bug detection techniques analyzed in Section II are quite effective for bug localization in small applications. For example, Eichinger et al. [8] evaluate their method against two known literature bug localization techniques ([6] and [7]) using a small dataset, generated using a `diff` implementation in `Java`. Although the effectiveness of the techniques is irrefutable, their efficiency is not thoroughly tested since the dataset is too small to resemble a real application. Indicatively, the size of the program is almost 2 pages of code, leading to call graphs of roughly 20 nodes after the reduction step.

A much more realistic dataset was used in [1]. The dataset was generated using the source code of `daisydiff` [22], a `Java` application that compares `html` files. `daisydiff` provides a more suitable benchmark since it has almost 70 files with 9500 lines of code. Thus, concerning scalability in a real application, that dataset is certainly sufficient. In this work, we have further extended the `daisydiff` dataset to test more thoroughly the scalability and the effectiveness of our methodology. We have planted 6 different bugs in the code of the 1.2 version of `daisydiff`, which are shown in Table II.

TABLE II. Planted Bugs

Bugs	Description	# Functions
1	Wrong limit conditions (Forgot +1)	637
2	Missing AND condition (Forgot a < check)	737
3	Wrong condition (> instead of <)	777
4	Missing OR condition (Forgot a != check)	723
5	Missing 1 of 3 AND conditions (Forgot a == check)	756
6	Missing 1 of 2 AND conditions (Forgot a == check)	638

The table contains the description of each bug as well as the average number of unique functions that are called in the respective runs. The dataset covers common types of bugs, such as missing boolean conditions, wrong conditions and wrong limit conditions. As mentioned above, the aim of these bugs is twofold. Experiments on different bug cases shall provide insight on the effectiveness of our algorithms, while the scalability of our methodology will be evaluated on different scenarios. The initial bug-free version of the program and the six buggy versions were all run 100 times given different inputs. The dataset can be found online in [23].

V. Evaluation

This section presents the results of applying the algorithms to the dataset described in Section IV. Upon creating the traces, the executions were crosschecked against the correct versions of the traces to provide the correct and incorrect sets.

TABLE III. ELAPSED TIME (IN SECONDS) FOR THE DIFFERENT PHASES OF THE ALGORITHMS FOR THE SIMPLESELECTOR APPROACH

	pq-Grams							NoTED	ZhangShasha						
	5	10	15	20	25	30	35		5	10	15	20	25	30	35
Graph Parsing	16.71	6.57	6.45	6.38	6.35	6.34	6.86	8.94	6.86	6.39	6.40	6.46	6.42	6.48	6.41
Graph Reduction	3.43	3.20	3.21	3.18	3.19	3.17	3.18	4.15	3.18	3.18	3.14	3.29	3.15	3.24	3.23
Dataset Reduction	70.84	70.49	70.53	70.03	69.72	69.64	69.74	0.00	162.50	162.31	161.54	162.38	162.00	162.78	160.75
Subgraph Mining	19.24	45.64	178.63	538.78	1391.44	1675.24	1973.72	24296.94	22.95	85.56	94.87	447.33	1319.32	714.84	1974.36
Ranking Calculation	0.51	1.75	6.87	17.75	42.61	75.08	97.47	2003.99	0.48	2.08	7.79	23.37	50.05	71.84	113.04
Total	110.73	127.65	265.69	636.12	1513.31	1829.47	2150.97	26314.02	195.97	259.52	273.74	642.83	1540.94	959.18	2257.79

TABLE IV. ELAPSED TIME (IN SECONDS) FOR THE DIFFERENT PHASES OF THE ALGORITHMS FOR THE STABLEMARRIAGE APPROACH

	pq-Grams							NoTED	ZhangShasha						
	5	10	15	20	25	30	35		5	10	15	20	25	30	35
Graph Parsing	6.78	6.40	6.33	6.28	6.27	6.38	6.29	8.94	6.38	6.46	6.32	6.45	6.36	6.38	6.39
Graph Reduction	3.19	3.23	3.17	3.14	3.17	3.18	3.24	4.15	3.14	3.19	3.17	3.16	3.15	3.15	3.21
Dataset Reduction	69.33	70.05	69.88	69.93	69.65	69.27	69.62	0.00	161.10	161.22	161.28	160.78	161.15	160.74	163.28
Subgraph Mining	35.51	111.15	334.84	322.36	884.47	998.60	827.78	24296.94	21.81	154.33	376.75	630.86	822.89	678.92	917.49
Ranking Calculation	0.57	3.63	12.07	21.77	32.34	38.72	42.68	2003.99	0.60	4.05	15.16	28.29	40.79	45.99	54.72
Total	115.38	194.46	426.29	423.48	995.90	1116.15	949.61	26314.02	193.03	329.25	562.68	829.54	1034.34	895.18	1145.09

TABLE V. ELAPSED TIME (IN SECONDS) FOR THE DIFFERENT PHASES OF THE ALGORITHMS FOR THE MAXLINKMAX APPROACH

	pq-Grams							NoTED	ZhangShasha						
	5	10	15	20	25	30	35		5	10	15	20	25	30	35
Graph Parsing	6.64	6.45	6.46	6.38	6.60	6.48	6.86	8.94	6.86	6.49	6.44	6.55	6.48	7.51	7.15
Graph Reduction	3.21	3.18	3.21	3.18	3.35	3.29	3.39	4.15	3.28	3.27	3.28	3.27	3.32	3.30	3.37
Dataset Reduction	70.79	70.66	71.66	70.85	74.41	74.35	74.79	0.00	169.19	168.89	168.29	168.24	168.29	168.56	168.53
Subgraph Mining	15.43	42.54	162.40	386.40	514.70	742.17	798.19	24296.94	19.13	226.84	379.05	412.76	844.61	1345.54	1439.16
Ranking Calculation	0.53	2.24	9.66	31.33	42.29	51.66	50.89	2003.99	0.54	3.07	12.95	28.67	55.16	75.43	74.60
Total	96.60	125.07	253.39	498.14	641.35	877.95	934.12	26314.02	199.00	408.56	570.01	619.49	1077.86	1600.34	1692.81

A. Experimental Setup

Several methods were implemented in order to test the validity of our dataset reduction hypothesis. An initial test was performed by applying the algorithm by Eichinger et al. [8] as discussed in Section II. However, due to performance limitations, subtree reduction (see Subsection II-B) could not be applied in such a large dataset. Thus, simple tree reduction is used in its place (refer to [1] for more details). In the conducted tests, the graph mining step is performed through the Parallel and Sequential Mining Suite (ParSeMiS) [24] implementation of CloseGraph, while the InfoGain algorithm of the ranking step was implemented using the Waikato Environment for Knowledge Analysis (WEKA) [25].

We implemented six more algorithms, accounting for all possible combinations of similarity metrics: ZhangShasha and pq-Grams implementations combined with all three call trace selection algorithms (SimpleSelector, StableMarriage and MaxLinkMax). For comparison reasons, all algorithms were implemented using the same libraries stated above. The experiments were conducted using 7 different values for the n parameter (5, 10, 15, 20, 25, 30, and 35) in order to explore its effect on performance and effectiveness. Concerning efficiency, the selection of these values for n is reasonable; the algorithms would be inefficient if they took into account more than 35 traces per set, i.e., 70 traces overall when the total number of traces is 100. In contrast, smaller values of n would compromise effectiveness. When few traces are retained per set, the bugs may not be distinguishable between these traces.

All experiments were performed using an 8-core i7 processor with 8 GBs of memory. The graph reduction, dataset reduction and subgraph mining steps were performed in parallel. Graph reduction was performed on 8 threads, where each thread performed simple tree reduction to a fragment of the dataset. The TED algorithms were applied in parallel using 4 threads (using more threads was impossible due to memory limitations) that calculated the TED for each correct-incorrect pair of the dataset. The call trace selection algorithms were executed sequentially. Finally, CloseGraph was executed using 8 threads, while the trace parsing and ranking steps were sequential.

B. Experimental Results

The algorithms are evaluated both in terms of effectiveness and performance. Concerning certain parameters, p and q of the pq-Grams approach were given the values 2 and 3 respectively, having little impact on performance and effectiveness, and CloseGraph was run with a 20% support threshold.

The performance results for all edit distance methods are shown in Tables III, IV and V for the SimpleSelector, StableMarriage and MaxLinkMax approaches respectively. These tables contain the average measurements for all six bugs of the dataset. Although the elapsed time required to localize a bug can be significantly different for two different bugs (e.g., bug 3 required 5 times more time than bug 1), the trend for all bugs is similar. Each table contains measurements for the

TABLE VI. RANKING POSITION AND PERCENTAGE OF FUNCTIONS TO BE EXAMINED TO FIND THE BUGS

		pq-Grams							NoTED	ZhangShasha						
		5	10	15	20	25	30	35		5	10	15	20	25	30	35
Bug 1	SS	7 (1.1%)	9 (1.4%)	31 (4.9%)	9 (1.4%)	8 (1.3%)	8 (1.3%)	8 (1.3%)	**5 (0.8%)**	7 (1.1%)	8 (1.3%)	9 (1.4%)	8 (1.3%)	9 (1.4%)	8 (1.3%)	8 (1.3%)
	SM	6 (0.9%)	8 (1.3%)	10 (1.6%)	8 (1.3%)	8 (1.3%)	8 (1.3%)	8 (1.3%)	**5 (0.8%)**	6 (0.9%)	8 (1.3%)	13 (2.0%)	12 (1.9%)	12 (1.9%)	12 (1.9%)	12 (1.9%)
	MLM	6 (0.9%)	8 (1.3%)	9 (1.4%)	8 (1.3%)	8 (1.3%)	8 (1.3%)	8 (1.3%)	**5 (0.8%)**	6 (0.9%)	8 (1.3%)	10 (1.6%)	8 (1.3%)	8 (1.3%)	8 (1.3%)	8 (1.3%)
Bug 2	SS	**5 (0.7%)**	**5 (0.7%)**	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	**5 (0.7%)**	**5 (0.7%)**	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)
	SM	**5 (0.7%)**	**5 (0.7%)**	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	**5 (0.7%)**	**5 (0.7%)**	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)
	MLM	**5 (0.7%)**	**5 (0.7%)**	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	**5 (0.7%)**	**5 (0.7%)**	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)	9 (1.2%)
Bug 3	SS	254 (32%)	252 (32%)	342 (44%)	27 (3.5%)	3 (0.4%)	**1 (0.1%)**	16 (2.1%)	17 (2.2%)	259 (33%)	253 (32%)	346 (44%)	355 (45%)	**1 (0.1%)**	**1 (0.1%)**	2 (0.3%)
	SM	168 (21%)	336 (43%)	341 (43%)	215 (27%)	3 (0.4%)	**2 (0.3%)**	16 (2.1%)	17 (2.2%)	246 (31%)	261 (33%)	353 (45%)	360 (46%)	9 (1.2%)	**2 (0.3%)**	4 (0.5%)
	MLM	256 (32%)	336 (43%)	6 (0.8%)	2 (0.3%)	**1 (0.1%)**	**1 (0.1%)**	**1 (0.1%)**	17 (2.2%)	256 (32%)	251 (32%)	3 (0.4%)	2 (0.3%)	2 (0.3%)	**1 (0.1%)**	**1 (0.1%)**
Bug 4	SS	270 (37%)	371 (51%)	**18 (2.5%)**	24 (3.3%)	29 (4.0%)	23 (3.2%)	51 (7.1%)	65 (9.0%)	268 (37%)	363 (50%)	27 (3.7%)	28 (3.9%)	35 (4.8%)	43 (5.9%)	29 (4.0%)
	SM	223 (30%)	391 (54%)	23 (3.2%)	23 (3.2%)	25 (3.5%)	24 (3.3%)	41 (5.7%)	65 (9.0%)	47 (6.5%)	399 (55%)	41 (5.7%)	**17 (2.4%)**	30 (4.1%)	35 (4.8%)	35 (4.8%)
	MLM	270 (37%)	35 (4.8%)	25 (3.5%)	25 (3.5%)	25 (3.5%)	25 (3.5%)	25 (3.5%)	65 (9.0%)	269 (37%)	33 (4.6%)	**20 (2.8%)**	**20 (2.8%)**	**20 (2.8%)**	**20 (2.8%)**	**20 (2.8%)**
Bug 5	SS	12 (1.6%)	6 (0.8%)	6 (0.8%)	6 (0.8%)	**5 (0.7%)**	**5 (0.7%)**	**5 (0.7%)**	**5 (0.7%)**	12 (1.6%)	6 (0.8%)	6 (0.8%)	6 (0.8%)	6 (0.8%)	6 (0.8%)	**5 (0.7%)**
	SM	19 (2.5%)	7 (0.9%)	6 (0.8%)	6 (0.8%)	6 (0.8%)	6 (0.8%)	6 (0.8%)	**5 (0.7%)**	19 (2.5%)	13 (1.7%)	7 (0.9%)	6 (0.8%)	6 (0.8%)	6 (0.8%)	6 (0.8%)
	MLM	12 (1.6%)	6 (0.8%)	6 (0.8%)	**5 (0.7%)**	**5 (0.7%)**	**5 (0.7%)**	**5 (0.7%)**	**5 (0.7%)**	12 (1.6%)	6 (0.8%)	6 (0.8%)	**5 (0.7%)**	**5 (0.7%)**	**5 (0.7%)**	**5 (0.7%)**
Bug 6	SS	6 (0.9%)	**3 (0.5%)**	4 (0.6%)	11 (1.7%)	9 (1.4%)	18 (2.8%)	15 (2.4%)	15 (2.4%)	22 (3.4%)	**3 (0.5%)**	**3 (0.5%)**	18 (2.8%)	9 (1.4%)	17 (2.7%)	15 (2.4%)
	SM	**3 (0.5%)**	**3 (0.5%)**	17 (2.7%)	18 (2.8%)	18 (2.8%)	18 (2.8%)	18 (2.8%)	15 (2.4%)	17 (2.7%)	**3 (0.5%)**	17 (2.7%)	15 (2.4%)	15 (2.4%)	15 (2.4%)	15 (2.4%)
	MLM	**3 (0.5%)**	**3 (0.5%)**	**3 (0.5%)**	18 (2.8%)	18 (2.8%)	18 (2.8%)	18 (2.8%)	15 (2.4%)	17 (2.7%)	17 (2.7%)	**3 (0.5%)**	15 (2.4%)	15 (2.4%)	15 (2.4%)	15 (2.4%)

*SS: SimpleSelector, SM: StableMarriage, MLM: MaxLinkMax

different values of n for each of the two similarity algorithms, pq-Grams and ZhangShasha, as well as the NoTED approach, which is the one not using any TED algorithm to reduce the size of the dataset.

In terms of total execution time, the proposed implementations clearly outperform the NoTED approach. In particular, even when n equals 35, the pq-Grams and ZhangShasha approaches require roughly 30 minutes using the SimpleSelector call trace selection algorithm. For the StableMarriage and MaxLinkMax algorithms, the respective value is even below 25 minutes. By contrast, the NoTED approach requires more than 7 hours in order to provide the final ranked list of functions. In relative terms, the proposed approaches are approximately 15 times faster than the NoTED approach.

Concerning all approaches, the mining step is indeed the most inefficient. Although ranking might also seem inefficient, its elapsed time depends mainly on the output of the mining step. Concerning the graph reduction step, simple tree reduction performs quite efficiently. The difference between the execution times of the tree edit distance algorithms pq-Grams and ZhangShasha is quite significant since the former is twice as faster as the latter. However, their contribution to the total execution time is rather insignificant with respect to the mining step. Using either of the proposed TED techniques, the choice of a call trace selection algorithm seems also irrelevant to the performance of the different implementations. Note, however, that since graph mining algorithms depend highly on the graphs that are given to them, dataset reduction could affect the measurement. Finally, although graph reduction techniques deviate from the scope of this paper, note that subtree reduction required many hours to reduce the graphs.

Table VI provides effectiveness measurements for locating the six bugs, for all different algorithms. The value inside each each cell of the table indicates how many functions should the developer examine in order to locate the bug. This metric is created using the final ranking of the functions and identifying the position of the "buggy" function. Using the total number of functions, which is shown in Table II for each bug, the percentage of the program's functions that should be examined to locate the bug is also provided. This is given inside a parenthesis in the value of each cell.

Our approaches seem to perform not only closely, but also even more effectively than the NoTED approach. In particular, our approaches provide a better ranking for bugs 2, 3, 4, and 6. The effectiveness metrics for bugs 3, 4, and 6 are very promising; these bugs seem to be the most "difficult" to locate. They are localized ineffectively by the NoTED approach, requiring 17, 65, and 15 functions respectively to be examined in order to find them. Our approaches outperform these results for bugs 3 and 4, as long as n is large enough. Concerning the sixth bug, the localization is even more satisfactory; our algorithms manage to localize the bug even when the values of n are small. This is also true for bugs 1 and 2, where the bug is localized either almost as good as the NoTED approach (for bug 1) or even better (for bug 2). Finally, the results for the fifth bug are also quite encouraging since our algorithms perform no worse than the NoTED one.

Since the nature of each bug is complex, no safe assumption can be made concerning the effectiveness of any algorithm with respect to the type of each bug or the number of function calls. In other words, the effectiveness is highly dependent on the selected dataset. Removing or switching certain boolean conditions can lead to bugs that are very difficult to locate, such as bugs 3, 4, and 6, or to easier cases, such as bugs 2 and 5. This is expected since the structure of the call traces can be considerably altered by any bug. Notice, for instance, how bugs 5 and 6, though similar, result to traces with 756 and 638 (unique) function calls respectively. In any case, the planted bugs are quite indicative of those arising in realistic scenarios.

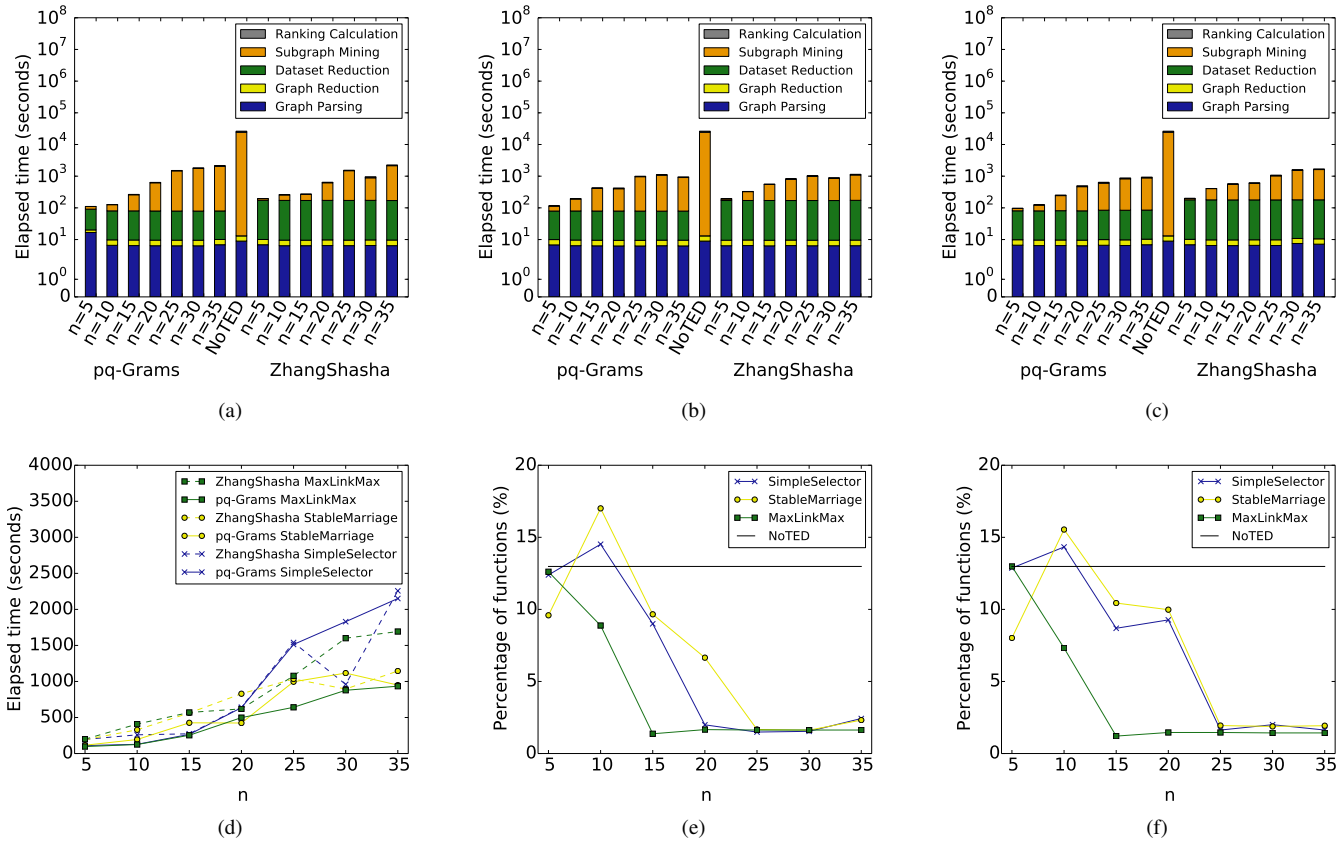


Figure 8. Average performance and effectiveness diagrams for the bugs of the dataset. Diagrams (a), (b) and (c) illustrate the performance for each phase of the algorithms in logarithmic scale versus the value of n (which denotes the number of traces retained from each of the two sets, correct and incorrect). The three diagrams correspond to the SimpleSelector, StableMarriage and MaxLinkMax algorithms. Diagram (d) depicts the total elapsed time of all combinations of the TED approaches, pq-Grams and ZhangShasha, with the call trace selection approaches, SimpleSelector, StableMarriage and MaxLinkMax. Diagrams (e) and (f) illustrate the percentage of functions to be examined in order to detect the bug versus n, for the pq-Grams and ZhangShasha approaches respectively.

Our conclusions regarding both effectiveness and performance are also confirmed by plotting the results, as in Figure 8. Figures 8a, 8b, and 8c illustrate the performance of our methodology for the SimpleSelector, StableMarriage and MaxLinkMax approaches respectively. Note that the vertical axis in these figures is in logarithmic scale in order to sufficiently illustrate the steps of the algorithms. As expected, performance is largely affected by the number of graphs taken into account, i.e., the n parameter. The impact of the number of graphs is better depicted in Figure 8d; the execution time of all approaches is high-order-polynomial with respect to consecutive values of n. This is expected since subgraph mining algorithms, such as CloseGraph, are largely affected by the size of the graphs and the size of the dataset.

Further analyzing Figure 8d, pq-Grams seems to execute faster than ZhangShasha for most values of n, regardless of which call trace selection algorithm is used. Peaks such as the one of the ZhangShasha with SimpleSelector approach are not totally unexpected since the performance of subgraph mining algorithms may be affected by numerous properties, such as the structure of the graph. In any case, useful conclusions can also be drawn for the performance of the three different call trace selection approaches. MaxLinkMax is both efficient and stable, indicating that it is robust and fits the problem better than the other two algorithms.

Concerning effectiveness, the impact of n is illustrated in Figures 8e and 8f, which depict the percentage of functions required to be examined versus n for the three call trace selection algorithms, and the NoTED approach. These figures correspond to the pq-Grams and the ZhangShasha approaches respectively. As shown in this figures, the effectiveness of our algorithms is indeed significant for large enough values of n. In specific, these average values indicate that our algorithms outperform the NoTED approach as long as n is larger than or equal to 15, while the MaxLinkMax approach outperforms it even for values larger than 10.

Given that n is the number of traces retained from the two sets (correct and incorrect), its impact on effectiveness is rather expected. In specific, when few traces are kept from each set (e.g., for n values lower than 15), the algorithms may not effectively isolate the bug since it may not be clearly distinguishable between these few traces. On the other hand, large values of n ensure that at least some of the retained traces will be highly relevant for isolating the bug. However,

since larger n values result also in larger execution times, it is preferable to select values near 25 or 30 where the algorithms exhibit high effectiveness while also being efficient.

Figures 8e and 8f illustrate the relative effectiveness of the call trace selection techniques. In particular, the MaxLinkMax approach outperforms the other techniques in terms of both effectiveness and stability. The algorithm seems to converge to satisfactory values faster than its opposing techniques. In Figure 8e, the percentage of functions to be examined for the MaxLinkMax approach drops below 2% for all n values that are lower than or equal to 15. The SimpleSelector and StableMarriage approaches require values lower than or equal to 20 and 25, respectively, to exhibit similar effectiveness. The results for the ZhangShasha algorithm are even more characteristic, with the SimpleSelector and StableMarriage requiring both at least 25 call traces per set in order to approach the effectiveness achieved by MaxLinkMax when the number of kept call traces is at least 15.

As a result of the above analysis, the effect of the call trace selection algorithm on localizing the bugs is quite significant. The MaxLinkMax algorithm was actually expected to prevail since it is the most well adapted algorithm to the problem at hand; intuitively, determining the unique graph pairs with the maximum pair weights should provide satisfactory results. Allowing duplicates and disregarding weights, as done by SimpleSelector and StableMarriage respectively, can be seen as drawbacks, therefore, leading to suboptimal solutions. As expected, however, these algorithms perform satisfactorily for large n values, since useful trace information is then retained.

The relative effectiveness among the approaches using pq-Grams and the three ones using ZhangShasha may seem slightly surprising. The approximate pq-Grams implementations seem more stable and more effective than their exact ZhangShasha counterparts. However, the difference between the MaxLinkMax approaches is not significant, and the effectiveness for the other two call trace selection algorithms differs only for small values of n. This indicates that both pq-Grams and ZhangShasha provide satisfactory results, as long as the call trace selection algorithm properly utilizes the weight values provided by them.

VI. Conclusion and Future Work

Current approaches in the field of dynamic bug detection suffer from scalability issues. In this paper, we have expanded on previous work [1], further testing the effect of reducing the size of the call trace dataset on both performance and effectiveness. With support from the experimental results of Subsection V-B, we argue that our approaches exhibit considerable improvement on the current state-of-the-art, even more so since they are tested on a realistic dataset.

Concerning the dataset reduction step, both TED algorithms are effective in terms of finding relative edit distances between graphs. Thus, the main scope of this work focused on exploring different possibilities for selecting the most useful call traces. We provided three call trace selection algorithms and explored how they affect the effectiveness of our methodology. The third algorithm we proposed, MaxLinkMax, proved to be the most stable and effective, since it outperformed all other implementations, while also being highly efficient.

Although the field of locating non-crashing bugs is far from exhausted, we argue that our methodology provides an interesting perspective on the problem. In specific, this work indicates that analyzing the call trace dataset to isolate useful traces yields quite promising results. Hence, future research includes further exploring the applicability and effectiveness of our techniques on different datasets. Furthermore, the parameters of our techniques (mainly the number of retained traces) could be optimized or automatically derived for each dataset. Finally, further analysis of the newly defined problem of call trace selection with respect also to the subgraph mining step could lead to more effective solutions.

References

[1] T. Diamantopoulos and A. Symeonidis, "Towards Scalable Bug Localization using the Edit Distance of Call Traces," in Proceedings of the Eighth International Conference on Software Engineering Advances (ICSEA), Oct 2013, pp. 45–50.

[2] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan, "Bug Isolation via Remote Program Sampling," SIGPLAN, vol. 38, no. 5, 2003, pp. 141–154.

[3] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable Statistical Bug Isolation," SIGPLAN, vol. 40, no. 6, 2005, pp. 15–26.

[4] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff, "SOBER: Statistical Model-based Bug Localization," SIGSOFT Softw. Eng. Notes, vol. 30, no. 5, Sep. 2005, pp. 286–295.

[5] M. Renieris and S. Reiss, "Fault Localization with Nearest Neighbor Queries," in Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE), 2003, pp. 30–39.

[6] C. Liu, X. Yan, H. Yu, J. Han, and P. S. Yu, "Mining Behavior Graphs for "Backtrace" of Noncrashing Bugs," in Proceedings of the 2005 SIAM International Conference on Data Mining, 2005, pp. 286–297.

[7] G. Di Fatta, S. Leue, and E. Stegantova, "Discriminative Pattern Mining in Software Fault Detection," in Proceedings of the 3rd International Workshop on Software quality assurance (SOQUA), 2006, pp. 62–69.

[8] F. Eichinger, K. Böhm, and M. Huber, "Mining Edge-Weighted Call Graphs to Localise Software Bugs," in Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I, 2008, pp. 333–348.

[9] F. Eichinger, C. Oßner, and K. Böhm, "Scalable Software-Defect Localisation by Hierarchical Mining of Dynamic Call Graphs," in Proceedings of the 2011 SIAM International Conference on Data Mining, 2011, pp. 723–734.

[10] S. Nessa, M. Abedin, W. E. Wong, L. Khan, and Y. Qi, "Software Fault Localization Using N-gram Analysis," in Proceedings of the Third International Conference on Wireless Algorithms, Systems, and Applications. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 548–559.

[11] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan, "Identifying Bug Signatures Using Discriminative Graph Mining," in Proceedings of the Eighteenth International Symposium on Software Testing and Analysis. New York, NY, USA: ACM, 2009, pp. 141–152.

[12] X. Yan and J. Han, "gSpan: Graph-Based Substructure Pattern Mining," in Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM), 2002, pp. 721–724.

[13] S. Nijssen and J. N. Kok, "A Quickstart in Frequent Structure Mining Can Make a Difference," in Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, NY, USA: ACM, 2004, pp. 647–652.

[14] Y. Chi, Y. Yang, and R. R. Muntz, "Indexing and Mining Free Trees," in Proceedings of the Third IEEE International Conference on Data Mining (ICDM), 2003, pp. 509–512.

[15] X. Yan and J. Han, "CloseGraph: Mining Closed Frequent Graph Patterns," in Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, pp. 286–295.

[16] K.-C. Tai, "The Tree-to-Tree Correction Problem," Journal of the ACM, vol. 26, no. 3, Jul. 1979, pp. 422–433.

[17] K. Zhang and D. Shasha, "Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems," SIAM J. Comput., vol. 18, no. 6, Dec 1989, pp. 1245–1262.

[18] N. Augsten, M. Böhlen, and J. Gamper, "Approximate Matching of Hierarchical Data Using pq-Grams," in Proceedings of the 31st International Conference on Very Large Data Bases, 2005, pp. 301–312.

[19] D. Gale and L. S. Shapley, "College Admissions and the Stability of Marriage," American Math. Monthly, vol. 69, no. 1, 1962, pp. 9–15.

[20] D. Gusfield and R. W. Irving, The Stable Marriage Problem: Structure and Algorithms. Cambridge, MA, USA: MIT Press, 1989.

[21] M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh, "Stability, Optimality and Manipulation in Matching Problems with Weighted Preferences," Algorithms, vol. 6, no. 4, 2013, pp. 782–804.

[22] "daisydiff: A Java Library to Compare HTML Files," [retrieved May, 2014]. [Online]. Available: http://code.google.com/p/daisydiff/

[23] "Software & Algorithms, ISSEL," [retrieved May, 2014]. [Online]. Available: http://issel.ee.auth.gr/software-algorithms/

[24] "ParSeMiS: The Parallel and Sequential Graph Mining Suite," [retrieved May, 2014]. [Online]. Available: https://www2.cs.fau.de/EN/research/zold/ParSeMiS/

[25] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," SIGKDD Explor. Newsl., vol. 11, no. 1, Nov 2009, pp. 10–18.

On Exploiting Passing and Failing Test Cases in Debugging Hardware Description Languages

Bernhard Peischl
Softnet Austria
Graz, Austria
bernhard.peischl@soft-net.at

Naveed Riaz
College of Comp. Science and IT
University of Dammam,
Dammam, Saudi Arabia
nrmohammed@ud.edu.sa

Franz Wotawa
Institute for Software Technology
Graz University of Technology
Graz, Austria
franz.wotawa@ist.tuGraz.at

Abstract - **In this manuscript, we outline how to use test suites for software debugging of hardware description languages. We propose an algorithmic improvement for dealing with numerous failing test cases and show how to exploit passing test cases in terms of a technique called filtering. We report on results obtained on a well-known benchmark suite. The results clearly show that both passing and failing tests are capable of increasing the diagnoses accuracy in the field of software debugging.**

Model-based debugging; software debugging; debugging of hardware description languages; fault isolation.

I. INTRODUCTION

This article is an extension to previous research work [1] and reports on recent results in software debugging of Verilog designs. In contrast to the Very High Speed Integrated Hardware Description Language (VHDL) [2], Verilog [3] has a formal semantics and thus is amendable to research in verification and debugging, e.g., its synthesis semantics is formally specified in Gordon [4]. Whereas VHDL is a strongly and richly typed language, Verilog is a weakly and limited typed language [5].

Most of the research in verification deals with the detection of faults and does not address the fact that debugging involves locating and correcting the fault. In detecting faults (software/hardware testing), we make use of numerous test cases. In the recent past, numerous test cases have been employed for localizing faults, e.g., in terms of employing spectrum-based diagnosis [6, 7, 8, 9, 10].

Spectrum-based techniques, however, allow for logical reasoning at the level of dependencies and do not consider the semantics of the language in terms of value-level models [11, 12]. Our work exploits synthesis semantics and makes use of test suites. This article shows that there is solid empirical evidence that taking into account test suites improves the fault localization in HDLs considerably.

Over the last 25 years, the Artificial Intelligence (AI) community has developed a framework for system diagnosis called model-based diagnosis (MBD). This framework covers a broad range of capabilities including the isolation of faulty components and the handling of multiple fault locations [13, 14]. A specific problem solving system is automatically generated by applying task-specific, but domain-independent problem solving algorithms (e.g., Greiner et al.'s algorithm [15]) to the system model. Harnessing these techniques in software engineering tools may help to master the development of complex circuits and software-enabled systems. The state of the art in this field can be characterized by prototypes that are starting to become part of industrial applications.

In this article, we extend previous work [1, 11, 12] in the field of debugging Hardware Description Languages (HDLs) by (1) introducing an iterative version of Greiner et al.'s hitting set algorithm and (2) presenting an empirical evaluation of the impact of passing test cases. Both aspects contribute to further establish AI-based techniques in the software engineering field.

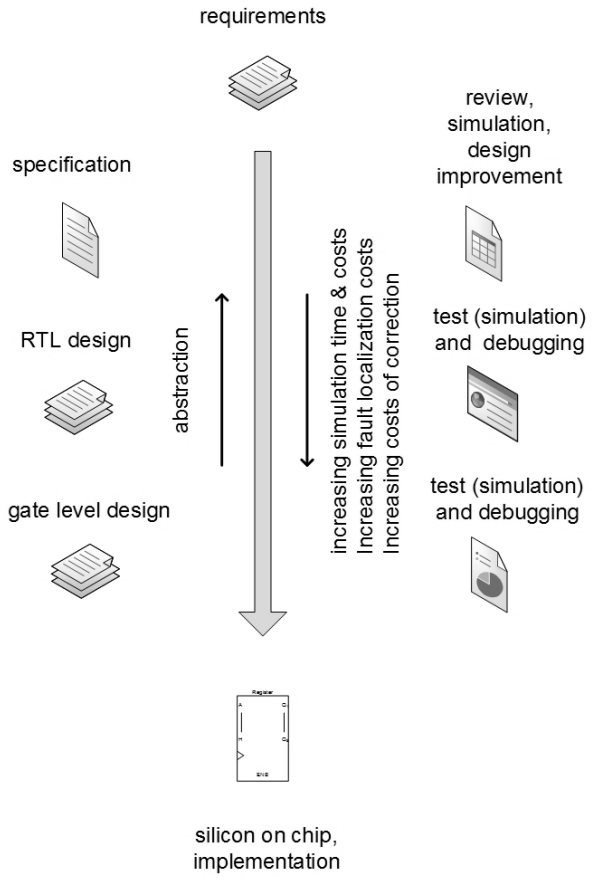


Figure 1: Design process with HDLs.

II. SIMULATION, TEST AND DEBUGGING

Figure 1 outlines an overview of the hardware design cycle employing the Verilog HDL. The designer starts with an initial specification that primarily captures the functional requirements for the circuit being designed. Usually, this is followed by a detailed design on the register transfer level (RTL). Both designs are executable and thus are amenable to automated verification. In general, the RTL design is verified very thoroughly in terms of testing and various other analysis techniques, e.g., hazard analysis. Since there is a fixed window for start of production, these verification steps typically are conducted under time pressure and thus the time for debugging – detecting, localizing, and repairing the misbehavior – becomes a key performance indicator.

Typically, the design process iterates through several steps: design and programming is followed by a simulation of the circuit. The outcome of the simulation is compared to the specification, that is, it is checked whether the waveform traces on a higher abstraction level (the specification) deviate from the waveforms obtained from the test run on the RTL level. Previous research work, carried out in the VHDL domain, gives an intuitive understanding on how to leverage model-based diagnosis for fault localization in HDL designs (see www.ist.tugraz.at/staff/peischl/HDLDebugging.wmv).

Moreover, to reduce costs and the time to market, it is of utmost importance to detect the faults as early as possible. Thus, as testing is a viable economical technique to assure functional correctness, testing is also subject of numerous research and innovation projects. However, in order to resolve a bug, it is equally important to localize and finally remove the fault. In terms of process maturity this is captured by the defect backlog metrics (which counts the number of removed bugs) rather than the defect arrival curve that captures solely the detection rate of faults [16]. In today's software/hardware engineering processes such key performance indicators often are made available by extracting data from the underlying development tools [17, 18] thus offering the potential to quantify the effect of introducing fault isolation tools.

According to a study conducted at IBM Haifa, 50 to 80 percent of the overall development is attributed to verification activities including localization and correction [19]. Thus, particularly under local or temporal separation of the design and the test team, the automation of fault localization (and correction) is a sustainable topic for ongoing and future R&D work as it contributes to make the development process more efficient.

III. DEBUGGING SEQUENTIAL VERILOG DESIGNS

In contrast to our previous research dealing with VHDL [12, 20, 21, 22] the semantics of Verilog has been analyzed rigorously, and thus provides the necessary theoretical underpinning in language semantics and circuit synthesis. Gordon [4] provides a formal description of various semantic interpretations of Verilog like event-semantics and trace-semantics. In event-semantics (which is the semantics employed for fine-grained simulations) the change of a variable necessitates the recalculation of depending procedures.

In contrast to that, the trace semantics of Verilog computes solely the quiescent values at the end of a simulation cycle. That is, trace semantics abstracts over transient states and computes the steady values at the end of the simulation cycle. For computing these quiescent values, each procedure is evaluated only once per cycle [4]. Procedures are evaluated in a certain order such that a procedure is not evaluated until all its driving procedures have been evaluated. In other words, a procedure's outputs are computed only when all its inputs are known (or can be computed). So we build up our representation of the design by starting with processes solely dependent on known inputs and variables (e.g., the primary inputs, including clock). Afterwards, the outputs of these processes are attached to the list of already known inputs and variables. This process continues until all the procedures in the design are levelized [22]. In this way, we build up a chain of procedures and their inputs and outputs, thus allowing one for an evaluation of all the variables used in the design at the end of the simulation cycle.

Synchronous sequential circuits change their states and output values at discrete instants of time, which are specified by the rising and falling edge of a clock signal. In electrical engineering, sequential circuits are often viewed as a sequence of connected combinational circuits. This can be done by selecting specific connections (e.g., one can use minimal-cut set computation [23] for identifying these connections) and splitting them in two separated connections. The output of a stage of a specific cycle is connected to the corresponding input of the next cycle. We have adopted the same idea for providing an appropriate debugging model for sequential designs. Our representation can be broken into two phases, one in which latches change state, and one in which all the combinational blocks are evaluated. We effectively break the design at latches by treating the outputs of the latches as they were inputs and inputs of the latches as they were outputs.

In our representation, we first identify variables that we have to synthesize into latches. By splitting these variables and treating them as additional inputs and outputs, we ensure that our representation remains acyclic. Then we levelize the graph according to the levelization strategy discussed above. Thus, we receive a sequence of procedures depicting the data flow from the given primary inputs to the primary outputs. Our next step is to unroll the sequential circuits to incorporate

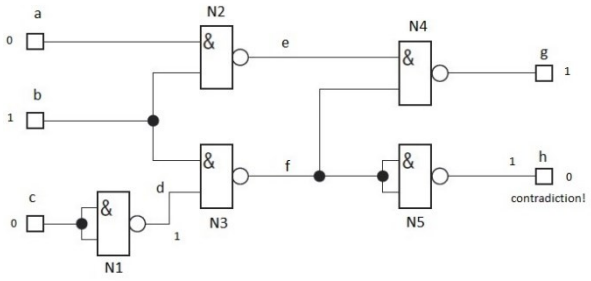


Figure 2: Illustration of a simple diagnosis problem.

multiple cycles (input sequence length). We assume that we know the number of unrollings to be performed in advance. After the levelization of all the procedures, we create the debugging model which represents our model at level 1 (cycle number 1). For every component *C*, we attach a timestamp *i* during the creation of the model to ensure a unique identification, where C_i represents the instance of component *C* at cycle *i*. Thus, we make n copies of every component involved, where n is the total number of cycles or unrollings. In this way, we create n number of instances for each component.

Diagnosis problem: A diagnosis problem considering circuit unrolling over *n* cycles is a triple *(SD, COMP,OBS)* where

1. $SD = \bigcup_{i=1..n} SD_i$ where SD_i is the system description for cycle i
2. $COMP = \bigcup_{i=1..n} C_i$ where C_i are the components in cycle i, and
3. $OBS = \bigcup_{i=1..n} OBS_i$ and OBS_i denote the observations in cycle *i*.

For every component of our model that is associated with the source code, we add an assumption $\neg AB(C)$. From a semantics point of view, this assumption denotes that the component C is assumed to work corretly. In other words this means it is assumed to be not abnormal. If we set this assumption to false, this means that the component is erroneous.

Example: Consider the digital circuit in Figure 2, which comprises five digital *NAND* gates, N_1 to N_5 and only a single cycle. We further assume that we have observed the following values on the digital circuit's inputs and outputs: *a=0*, *b=1*, *c=0*, *g=1*, and *h=0*. These values correspond to the observations *OBS*. The system description *SD* corresponds to the syntax and the semantics of the circuit (e.g., a constraint model, or horn-clause encoding of the circuit). Obviously, *SD* and *OBS* are contradictory. We can prove this by computing the values for every gate's outputs (and inputs). From *a=0* and *b=0*, we conclude that the output of gate N_2 becomes *1*. From *c=0* follows that the second input of gate N_3 must be *1*. This value together with *b=1* leads to *f=0*. Consequently, *h=1* contradicts the observed value for *h*. So, we know that something must be wrong and that the assumption that all components are working correct can no longer be valid.

The above given definition captures a diagnosis model for a single test case (of length *n*). Given this definition the diagnosis problem considering a test suite is given as follows, where we refer to the predicate *AB(C)* to denote abnormality of component *C* (correspondingly $\neg AB(C)$ refers to a correctly functioning component).

Diagnosis problem, test suite: Given a test suite comprising the test cases $TC_1, TC_2, \ldots, TC_k$. Let the system description SD_j be the system description considering test case TC_j and let C_i^j be the instance of component *C* at cycle *i* in test case number *j*. Correspondingly OBS_i^j denote the observations in cycle *i* of test case TC_j. The diagnosis problem *(SD*, COMP*, OBS*)* considering this test suite is given as follows:

1. $SD^* = \bigcup_{j=1..k} SD_j \cup \{\neg AB(C_0^j) \rightarrow \neg AB(C_1^j) \wedge \ldots \wedge \neg AB(C_n^j)\}$
2. $COMP^* = \bigcup_{j=1..k, i=0..n} C_i^j$
3. $OBS^* = \bigcup_{j=1..k, i=1..n} OBS_i^j$

IV. Iterative Computation of Diagnoses

In computing the diagnosis candidates we determine all inconsistent sub models (i.e., parts of the given design causing discrepancies). In the terminology of model-based diagnosis (MBD) these sub-models are referred to as conflicts. Since the assumption that all components of a conflict behave correctly causes the discrepancy, at least one of these components must be responsible for the misbehavior. Thus, once we have obtained all inconsistent sub models, for every component, we have to check, whether assuming this component to be abnormal allows one for getting rid of the given discrepancy in every sub model. We collect those assumption(s) that allow one for removing the given discrepancies and report the associated components as diagnosis candidates.

Recalling the previous definitions, the computation of diagnosis candidates is a consistency check for first-order sentences. In theory, one can compute diagnoses by generating all subsets Δ of *COMP* in increasing order of cardinality and checking whether $SD \cup OBS \cup \{\neg AB(C) \mid C \in COMP \setminus \Delta\}$ is consistent. Central to this algorithm is the concept of a contradictory sub model referred to as conflict in the classical MBD literature. A conflict for a diagnosis problem *(SD, COMP, OBS)* is a set $CO \subseteq COMP$ such that $SD \cup OBS \cup \{\neg AB(C) \mid C \in CO\}$ is contradictory. A conflict set is minimal iff no proper subset of it is a conflict set for *(SD, COMP, OBS)*. A set of conflicts is referred to as conflict-set $F=\{CO_1, CO_2, \ldots, CO_n\}$.
A conflict $CO = \{C_1, C_2, C_3, \ldots, C_k\}$ says that the assumption that all components are correct – that is,
$\neg AB(C_1) \wedge \neg AB(C_2) \wedge \neg AB(C_3) \wedge \ldots \wedge \neg AB(C_n)$ is true – is inconsistent with *SD* and *OBS*. However, *SD* together with *OBS* is consistent. Thus, the correctness assumptions $\neg AB(C_i)$ are responsible for the contradiction and must be altered to eliminate the conflict. This means that we must invert at least one of the $\neg AB(C_i)$ assumptions. If we now have more than one conflict, we must invert at least one (not necessarily different) assumption from every conflict. These inverted assumptions are a diagnosis because they resolve all

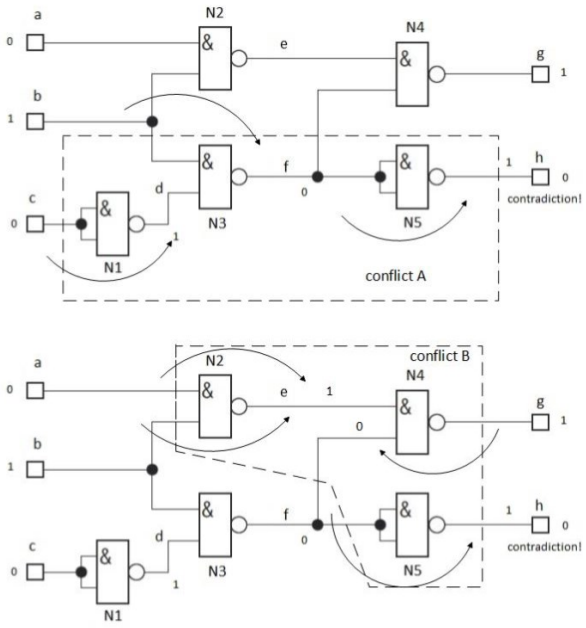


Figure 3: Example illustrating the computation of conflicts.

conflicts. So, a diagnosis is a set of components that, when assumed to behave incorrectly, leads to a consistent system state.

Continuing our example, we obtain two minimal conflicts. Figure 3 depicts them together with the computation of the contradiction values. There are two conflicts: *A*, whose components are N_1, N_3 and N_5, and *B*, whose components are N_2, N_4, and N_5. From this follows immediately that $\{N_5\}$ is a single-fault diagnosis candidate because $AB(N_5)$ resolves both conflicts *A* and *B* [12].

However, this rather inefficient brute-force approach does not work for debugging, as the number of components becomes huge. Reiter et al. [14] provide an algorithm for finding a set of minimal diagnoses and Greiner et al. [15] provide a correction to Reiter's proposal. Reiter et al. [14] and Greiner et al. [15] show how to efficiently compute diagnoses given a single conflict-set in terms of the hitting set algorithm.

The classical MBD literature is primarily focused on how to compute the diagnoses from a single conflict-set. However, in model-based software debugging, every failing test case results in one or several conflicts, i.e., a conflict-set. When considering several test cases TC_1, TC_2, TC_3, ..., TC_k, we obtain a conflict-set for every test case. The resulting set C of conflict-sets therefore is $C = \{F_1, F_2, F_3, ..., F_k\}$.

In theory we therefore can compute diagnoses by computing all minimal hitting sets for the union of the conflict-sets $\bigcup_0^k F_i$. However, in debugging HDLs, conflicts appear iteratively, e.g., first we execute test cases TC_1 (resulting in conflict-set F_1) and afterwards (when conflict-set F_2 becomes available) we execute a second test case TC_2 (resulting in conflict-set F_2). Following the classical literature, one can compute the diagnoses resulting from conflict-set F_1 and afterwards compute the diagnoses for the conflict-set $F_1 \cup F_2$. This results in building up the hitting set dag for the conflict-set F_1 twice as this dag needs to be built for test case TC_1 (conflict-set F_1) and for both test cases TC_1 and TC_2 (conflict-set $F_1 \cup F_2$).

In developing our automated debugging tool we managed to overcome this challenge by using an iterative variant of the original algorithm from Greiner et. al [15]. This algorithm answers the research question how to efficiently compute diagnoses in an iterative manner. Our algorithm consists of four main parts.

The procedure *Iterative_HS(C)* takes a set of conflict-sets $C=\{F_1, F_2, .., F_n\}$ and returns a dag. By collecting the edge labels *H(n)* at all nodes labeled with √ we can retrieve all (subset-minimal) diagnoses in increasing order of cardinality, i.e., all single-diagnoses can be retrieved prior to computing dual-fault diagnosis. For example, by retrieving all edge labels *H(n)* up to level three of the graph, we obtain all single- and dual-fault diagnoses. Note that the order in which the conflict-sets appear is determined by the availability of the test cases and the specific decision procedure for computing conflicts (e.g., the procedure given in [21]). Two different orderings of the same conflict-sets will result in different dags, however, from both dags we retrieve the same set of diagnoses.

The procedure *HSDAG(D, N, F)* is a modified version of the algorithm proposed in Geiner et. al. It differs from the original algorithm as it not only operates on the dag *D* and the conflict-set *F* but relies on an ordered set of nodes *N*. We use these nodes to control which nodes need to be modified in the case that the already existing dag (e.g., dag resulting from conflict-set F_1) becomes inconsistent with the new conflict being added (e.g., dag resulting from conflict-set F_1 is inconsistent with respect to conflict-set F_1 and F_2).

In order to determine these nodes, we use two further procedures. The procedure *Check_√ (D, F)* checks whether there are nodes marked with √ in the dag *D*, that according to the given conflict-set *F* are no longer valid. To establish the invariant of the algorithm, we need to label these nodes with the first set ∑ from *F* and store these nodes for later processing within *HSDAG*. The second procedure *Check_× (D, F)* checks whether there are closed nodes that need to be re-opened due to adding the conflict F. In this case, the respective node is re-opened and either labeled with the first set from F or marked with √. Again, we store this node for later processing as it might be subject of further pruning according to Greiner's algorithm.

Iterative_HS (C)

1. *Let DAG represent the growing dag. Let H(n) be the set of edge labels on the path in DAG from the root down to node n.*
2. *Generate a DAG_0 with root node n_0 with $label(n_0) = \Sigma$, where ∑ is the first set in conflict-set F_1 and $H(n_0) = \phi$.*
3. *Let N_0 be the nodes from DAG_0 in breath-first order*
4. *DAG = DAG_0; N=N_0.*
5. *For i=1 to |C|-1*
6. *DAG=HSDAG (DAG, N, F_i)*

7. N=Check_√ (DAG, F_{i+1}) ∪ Check_× (DAG, F_{i+1})
8. Return HSDAG(DAG, N, F_{i+1})

Comments:

1. Definition *DAG*, *H(n)* denotes the set of edge labels
2. The initial dag DAG_0 contains the root node n_0 labeled with the first element from conflict-set F_1, and two children
3. N_0 is the ordered set of nodes in DAG_0
4. Creation of the initial *DAG*
5. Iteration through all conflict-sets
6. Invoke *Check_√* and *Check_×* to retrieve those nodes from the *DAG* that need to be modified in order to be consistent with the passed conflict set F_i
7. For each conflict-set we invoke *HSDAG* explicitly given the set of nodes *N* that need to be modified
8. Finally, invoke *HSDAG* to return the pruned *DAG*

Check_√ (D, F)
1. R= ϕ
2. For all nodes $n \in D$ where label (n)= √ in breath-first order do
3. If there is $x \in F, H(n) \cap x = \phi$ then
4. $label(n)$ = ∑ where ∑ is the first element from F for which $\sum \cap H(n) = \phi$
5. R=R ∪ {n}
6. Return R

Comments:

1. Initially, the set of nodes to be processed is void
2. Traverse nodes labeled with √ in breath-first order
3. Check if node needs to be re-labeled
4. Label node n with the first element in *F* which is not in *H(n)*
5. Store node for further processing in *HSDAG*

Check_× (D, F)
1. R= ϕ
2. For all nodes $n \in D$ where label (n) = × in breath-first order do
3. If there is a node $n' \in D$ which is labeled by √ and $H(n') \subset H(n)$ then
4. If for all $x \in F, x \cap H(n) \neq \emptyset$ then label(n)= √
5. Otherwise, $label(n)$ = ∑ where ∑ is the first element from F for which $\sum \cap H(n) = \phi$
6. R=R ∪ {n}
7. Return R

Comments:

1. Initially, the set of nodes to be processed is void
2. Traverse nodes labeled with × in breath-first order
3. Check if node needs to be re-opened
4. Re-label node with √
5. Re-label node with the first element in F which is not in *H(n)*
6. Store node for further processing in *HSDAG*

HSDAG (D, N, F)
1. For all nodes $n \in N$ do
2. if for all $x \in F, x \cap H(n) \neq \emptyset$ then label(n)= √
3. Otherwise, $label(n)$ = ∑ where ∑ is the first element from F for which $\sum \cap H(n) = \phi$
4. If n is labeled by a set ∑, for each $\sigma \in \sum$, generate a new arc with $label(n) = \sigma$. This arc leads to a new node m with $H(m) = H(n) \cup \{\sigma\}$. The new node m in D will be processed after all nodes in the same generation as n have been processed.
5. [REUSE]
a. If there exists a node n' with H(n')=H(n) ∪ { σ } then generate a directed arc from n to n'. Hence n' will have more than one parent.
b. Otherwise, generate a new node m at the end of this σ-arc
6. [CLOSING]
If there exists a node n' labeled with √ where $H(n') \subset H(n)$, then set label(n') to × for closing n. A label is not computed for n nor any successor nodes generated.
7. [PRUNING]
If the set ∑ is to label a node and it has not been used previously then attempt to prune D as follows:
a. If there exists a node n' which has been labeled by a set $S \in F$ where $\sum \subset S'$, then relabel n' with ∑. For any α in $S' \setminus \sum$ the α-arc under n' is no longer allowed. The node connected by this edge and all of its descendants are removed, except for those with another ancestor which is not being removed.
b. Interchange $label(n)$ and $label(n')$
8. Return D

Comments:

1.-8. *HSDAG* computation according to Greiner [15]. The authors of [15] in detail describe strategies for closing, pruning and reuse of nodes.

The functions *Check_√ (D, F)* and *Check_× (D, F)* return those nodes in the dag that are not consistent with the new conflict-set F. Instead of applying the *HSDAG* procedure to the set $F_1 \cup F_2$, we apply it to conflict-set F_1 and identify those nodes that are not consistent with the new set $F_1 \cup F_2$. Afterwards, we alter, respectively, extend the dag obtained from conflict-set F_1 by applying the *HSDAG* procedure only for the identified nodes. Within *HSDAG*, the strategies for pruning, closing and re-using nodes are the same as proposed in Greiner et al. [15].

Example: Let F_1={{1,2,3}, {1,3}, {1,4}} and F_2={{1,4,5}, {3,4}, {1,2}} be conflict sets obtained from two test cases. Figure 4 represents the hitting set dag for F_1 and Figure 5 represents the corresponding hitting set graph for $F_1 \cup F_2$. In the following, we assume the existence of an ordered collection of conflict-set *Fi* for every test case *TCi* and show how to obtain the hitting set dag for the conflict-set *F1* ∪ *F2*.

Note that it is not necessary to compute all the conflicts for a given test case in advance, rather the algorithm allows one for computing the conflicts on demand. Furthermore, the algorithm allows one for computing the hitting sets in order of increasing cardinality. Thus, we can stop the computation of diagnoses once a specified depth has been reached (e.g., we retrieve solely single and dual-fault diagnosis by stopping computation at depth level 3). In the following, we illustrate the iterative variant of Greiner's hitting set algorithm that takes the set of conflict sets as an argument.

According to our algorithm, we have to build up the hitting set DAG_0. This initial dag consists of a node n_0 with two edges and $H(n_0)=\phi$. We continue with the computation of the hitting set dag $HSDAG_{F1}$ for conflict-set F_1 by invoking

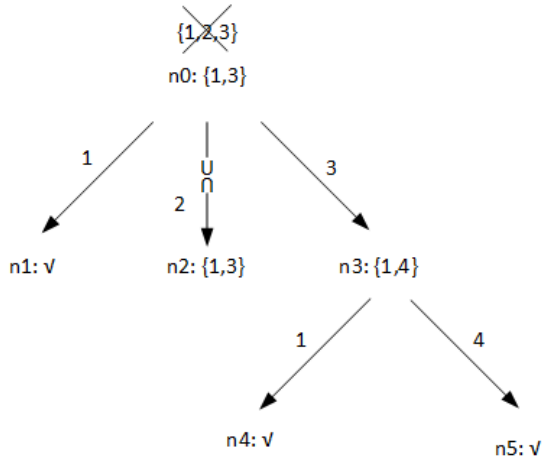


Figure 4: Hitting set dag HSDAGF1 for the conflict-set F1

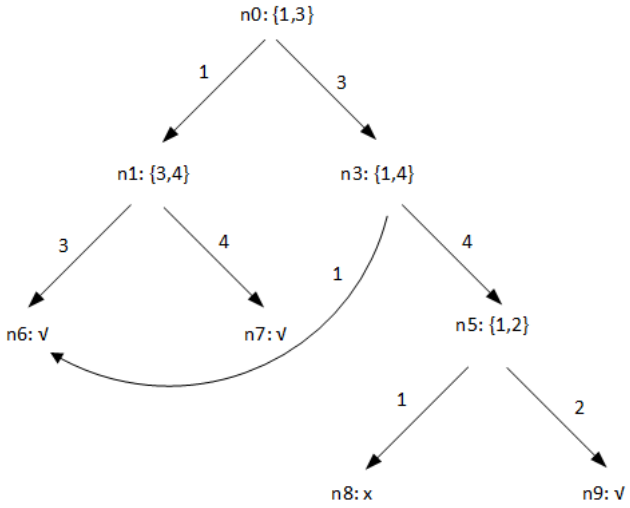


Figure 5: Hitting set for the union of the set $F_1 \cup F_2$

$HSDAG(DAG_0, N_0, F_1)$, where the collection N_0 represents the nodes from DAG_0 in breath first order.

This step corresponds to applying the algorithm as proposed by Greiner et al. [15].

As specified in the algorithm, in line 7, we determine those nodes in $HSDAG_{F1}$ that need to be recalculated as their labeling in no longer consistent with the extended conflict $F_1 \cup F_2$. As illustrated in Figure 2 (we used circular arcs to denote pruning of node n_2) the nodes n_1, n_5 (*Check_ ✓)* and node n_4 (*Check_ ×)* are the potential candidates for re-labeling or re-opening. After having executed *Check_ ✓* and *Check_ ×* we invoke *HSDAG* (DAG_1, N_1, F_2) to finally obtain $HSDAG_{F1UF2}$. This dag is consistent with the conflict-set $F_1 \cup F_2$. Figure 5 illustrates the final dag from which we retrieve the hitting sets *{1,3}, {1,4}* and *{2,3,4}* as the set of diagnoses.

V. EXPLOITING PASSING TEST CASES

Since passing test cases do not cause a logical contradiction, we do not obtain conflicts from passing test cases. However, passing test cases contribute in isolating faults in two ways.

First, we need passing test cases to bring the program into a state, in which another (failing) test case can reveal a misbehavior. In general, to exhibit misbehavior, sequential designs need to traverse a chain of intermediate states. In each of these states, the circuit does not exhibit erroneous behavior, and thus passing test cases contribute to finally reach a state in that the circuit exhibits misbehavior.

Second, as the different instances of our components behave independently, as we create an independent component for every unfolding of the circuit. We can use passing test cases to incorporate the notion of deterministic components into our debugging model. To illustrate the potential of using passing test cases to locate the root cause for detected misbehavior we continue with a simple example.

TABLE I: ASSUMPTIONS, AND TEST CASES FOR OUR RUNNING EXAMPLE.

assumption	in1	in2	out	inter	verdict
AB(not), ¬ AB(exor)	1	0	1	0	fail
AB(not), ¬ AB(exor)	0	0	1	1	pass

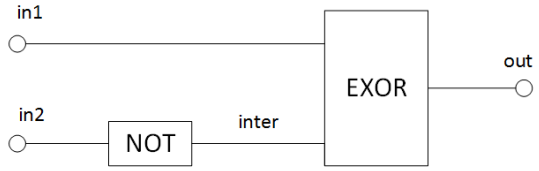


Figure 6: Part of a circuit as our running example.

As a part of a circuit, Figure 6 illustrates an exclusive or and a not gate together with a passing and failing test case for this circuit. We further assume that the circuit is faulty, that is, our test suite has identified misbehavior and we obtain both components (the *EXOR* and the *NOT* gate) as possible diagnosis candidates.

Suppose we have the test cases given in Table I. Considering the first (failing) test case in the first line, and assuming the *NOT* gate to be abnormal but the *EXOR* gate correct, we can deduce that variable *inter* becomes *0*. However, under the same assumption, the passing test case in line 2, forces variable *inter* to become *1*. We immediately see that the *NOT* gate is required to map the variable *inter* to *0* and to *1* for the same input value in_2=0. Obviously, no deterministic component can fulfill this requirement. Thus, the *NOT* gate can no longer be considered as a valid diagnosis candidate. To our best knowledge, the authors of [24] were the first who used this idea for discriminating diagnosis candidates. Unfortunately, the article gives no further insights whether the technique can be employed in practice as the authors do not provide an empirical evaluation to evaluate scalability and the improvement potential with respect to accuracy.

In the following, we propose an extension to that which – under absence of structural faults – allows one for taking advantage of passing test cases. As passing test cases do not yield to additional conflicts, we capture their specific information about diagnoses in terms of Ackermann constraints [25]. By adding these consistency constraints we incorporate the fact that the same combination of input values applied to a deterministic component *C* produces the same output for every instance of *C*. This allows one for exploiting the many test cases that do not reveal a fault. The system description with Ackermann constraints SD_A is given as follows:

System description with Ackermann constraints: Let *TC* be a set of test cases form a test suite *TC*, let $in(C_i) = \{ i_{Ci}^1, \ldots, i_{Ci}^m \}$ denote the inputs of component C_i, let $out(C_i) = \{ o_{Ci}^1, \ldots, o_{Ci}^n \}$ denote the outputs and let *SD** denote the system description of a diagnosis problem considering a test suite.

The system description with Ackermann constraints SD_A is given by,

$$SD_A = SD^* \cup CON_A,$$
$$CON_A = \neg AB(C_i) \wedge \forall_{l=1}^{m} i_{ci}^l = i_{cj}^l \rightarrow \forall_{p=1}^{n} o_{ci}^p = o_{cj}^p$$

where, $i \neq j$ and *i,j* denote indices of the test cases.

As we will show in the next section, Ackermann constraints increase the complexity of the model considerably. Therefore, we used a post processing technique proposed by the authors of [26]. As shown at the end of this section filtering allows one for iteratively applying the Ackermann constraints to the obtained diagnoses. Instead of compiling the constraints into the debugging model, we apply the constraints in terms of a dedicated post-processing phase.

Filtering refers to discarding certain diagnoses by taking advantage of further test cases TC_i. A diagnosis Δ states that $\Delta \cup SD \cup TC_i \cup \{\neg AB(C) \mid C \in COMP \setminus \Delta\}$ is consistent. This implies that there is a replacement, that is, there exists a function *replace(C)* for every component $C \in \Delta$ that allows one for repairing the program for the given test case. The function *replace(C)* allows one for producing the correct output values for the considered test case. However, considering a test suite such a replacement does not exist for all test cases in the test suite TC necessarily. Since all components $COMP \setminus \Delta$ are assumed to behave correctly, we can compute the input values *in(C)* and *out(C)* for every component *C* from Δ (employing forward propagation). According to this computed input/output relation, the component *C* may be required to map the same input- to different output values. This corresponds to an inconsistency and the specific diagnoses *AB(C)* is not repairable wrt. the specific test case. As there is no function *replace(C)* as stated previously, the component *C* can be removed from the set of diagnosis candidates. In this vein, we evaluate the Ackermann constraints in an iterative way by checking for different input values for a certain output value.

Algorithm 1 (Filtering): Let Δ denote a set of diagnosis candidates and let TS be a test suite.

1. *For all* $D \in \Delta$ *do*
2. *For all test cases* $TC_i \in TC$ *do*
   a. *Let* i_{Di} *denote the input values and let* o_{Dj} *denote the output values of component D by assuming* $AB(D) \wedge \{\neg AB(C) \mid C \in COMP \setminus D\}$
   b. *If there exits i,j,* $i \neq j$, *such that* $i_{Di} = i_{Dj} \wedge o_{Di} \neq o_{Dj}$ *then remove D from* Δ
3. *Return* Δ

Claim: Algorithm 1 applies the Ackermann constraints CON_A to a set of single-diagnosis candidates.

After applying Algorithm 1 to the set of single-fault diagnosis candidates, there is no component *D* at which we obtain different input values for a certain output value. Thus, we

conclude, that

1. $\neg\exists i,j,i\neq j \bullet (\forall_{l=1}^{m} i_{Di}^{l} = i_{Dj}^{l}) \wedge (\forall_{p=1}^{n} o_{Di}^{p} \neq o_{Dj}^{p})$
2. $\forall i,j,i\neq j \bullet \neg(\forall_{l=1}^{m} i_{Di}^{l} = i_{Dj}^{l}) \vee (\forall_{p=1}^{n} o_{Di}^{p} = o_{Dj}^{p})$
3. $\forall i,j,i\neq j \bullet (\forall_{l=1}^{m} i_{Di}^{l} = i_{Dj}^{l}) \rightarrow (\forall_{p=1}^{n} o_{Di}^{p} = o_{Dj}^{p})$

Thus, Algorithm 1 imposes the Ackermann constraints on the set of single-fault diagnosis candidates. For our evaluation of the approach we therefore took advantage of the filtering algorithm previously presented.

VI. Practical Experiences and Evaluation

Our evaluation and practical experiences answer two research questions. First, we strive to quantify the impact of exploiting passing test cases for debugging. This is done by referring to a former experiment [11] and comparing these results with our novel results considering passing test cases. Second, we evaluated the running times of the algorithm proposed herein. For both research questions, we rely on the ISCAS'89 benchmark suite [27].

We conducted our experiments on a Dell Power Edge 1950 II - 2x Quad Core with 2.0 GHz and 10GB of RAM. For computing diagnoses we relied on the extension of Reiter's algorithm described herein. Note that, for the efficient computation of diagnoses, we convert the rules given in the previous sections into a specific Horn-like encoding [21]. As the computation of conflict sets is a time critical issue, the (minimal) conflict sets are computed according to the procedure explained in [21]. The diagnosis engine and the proposed extension are implemented in the Java programming language.

Our debugging tool parses the Verilog code, builds up the model as described in this article and converts a test suite to the logical representation. Afterwards, the tool computes diagnosis candidates in increasing order of cardinality and visualizes the results by highlighting the corresponding statements, expressions or operators.

A. Time Complexity of Computing Diagnoses

For our empirical evaluation we use a Horn-like encoding of the rules presented herein. By relying on this encoding we make use of an efficient procedure to compute all minimal conflicts [21]. From the obtained conflicts we retrieve diagnoses by computing the minimal hitting sets in increasing order, where for practical purposes, primarily single- and dual-fault diagnoses are of interest. In general, searching for all diagnoses has a worst time complexity of the order $O(|MODES|*|COMP|^{s})$, where $|MODES|$ is the number of fault modes, $|COMP|$ is the number of components and s is the maximal size of the diagnoses [23]. Since we use two fault modes ($AB(C)$ and $\neg AB(C)$) and search for single and double fault diagnoses, our worst time complexity is of the order $O(|COMP|^{2})$. Note that we consider the components in every cycle as independent and thus the number of components increases with the length of the test case. However, the average running time complexity is much better because diagnoses with smaller cardinality (particularly single-fault diagnoses) are more likely than higher order diagnoses. For example, finding all single diagnoses is of order $O(|COMP|)$ assuming the decision procedure can be executed in unit time.

B. Generation of Test Suites

We obtained the test suite by injecting a single-fault (respectively a dual-fault for the second series of experiments) into the RTL design. Afterwards, we identified the faults in terms of running a simulation until we obtained five test cases revealing the introduced fault. The faults are introduced in a random way by picking a statement from every circuit and replacing this statement by another statement. That is, for every circuit, we replaced an arbitrary statement with a structurally equivalent statement (same no. of input parameters). For example, in a specific circuit we randomly selected a *NOR* statement and replaced it by an *AND* statement. Further, we implicitly removed/added negations as we substituted a logical statement by the negated counterpart (e.g., *NAND* by *AND* or vice versa). These error types are not necessarily complete w.r.t. functional errors, but as they are believed to be common in the design process, we capture the most common scenarios [28]:

- Mistakenly replacing one gate/statement by another gate/statement with the same number of inputs.
- Incorrectly adding or removing a gate/statement.

All empirical evaluations are conducted on the Verilog RTL version of the ISCAS'89 benchmark suite [27]. Further, the gate-level representations of the ISCAS'89 benchmarks have been used to obtain the correct waveform traces since our simulator allowed only for simulation of gate-level circuits.

Regarding all experiments, we verified that the injected fault (the root cause) is among the retrieved diagnosis candidates.

C. Empirical Evaluation

In our experimental setting, we assumed that an engineer only knows the correct values of the primary inputs for every simulation cycle and the outputs at the end of the final simulation cycle. That is, specified information captures the primary inputs v_{in} and their corresponding values val_{in} for every instant of time $t=1..n$, (v_{in}, val_{in}, t), together with the primary outputs and the corresponding value at time n, (v_{out}, val_{out}, n). The observations are given in terms of the primary input variables for every cycle and the primary output variables at the end of the simulation cycle (i.e., at time point n, where n is the length of the test case). Table II lists the number of primary inputs, primary outputs, and the number of gates and D-type flip-flops for the circuits we considered in our experiment. The last column is not published in Brglez et al. [27] but lists the number of lines of code in the source code representation of the Verilog RTL design.

In our first experiment, we evaluate the discrimination capability of the software debugger with an increasing number of failing test cases. Figure 7 summarizes the number of obtained single-fault diagnoses for a part of the ISCAS'89 benchmark suite.

TABLE II: STRUCTURAL CHARACTERISTICS OF THE PROGRAMS FOR THE EMPIRICAL EVALUATION.

circ. name	no. prim. inp.	no. prim. outp.	no. D-type flip-flops	no. gates	no. lines
s208	11	2	8	96	240
s349	9	11	15	161	545
s382	3	6	21	158	544
s386	7	7	6	159	508
s444	3	6	21	181	602
s510	19	7	6	211	660
s526	3	6	21	193	634

For every circuit we randomly introduced a single fault and computed all single-fault diagnoses using the algorithm introduced in Section IV. Regarding this experiment, the introduced fault always was among the set of retrieved single-fault diagnoses.

The experiment is of practical relevance as in practice, engineers have a limited amount of failing test cases only (in our case up to five) and numerous passing test cases. The failing test cases relate the primary inputs to the (quiescent) value of the primary outputs. Our experiment does not assume any intermediate values to be known (e.g., the expected temporal response for the primary outputs). We solely rely instead on the input values for every cycle and the expected value of the primary outputs at the end of the simulation.

Figure 7 underpins the findings discussed in previous research articles as the number of single-fault diagnoses being obtained depends on both, the concrete test case and the structural complexity of the program. With an increasing number of failing tests we can considerably reduce the number of obtained single-fault diagnoses.

Afterwards, we repeated a similar experiment but in addition applied the filtering algorithm to exploit the numerous passing test cases for debugging. Figure 8 illustrates the improvement we gained from applying these test cases together with the failing test cases. In the figure, no passing test case (0) refers to using exactly five failing test cases. In addition, we applied the filtering procedure by using up to four

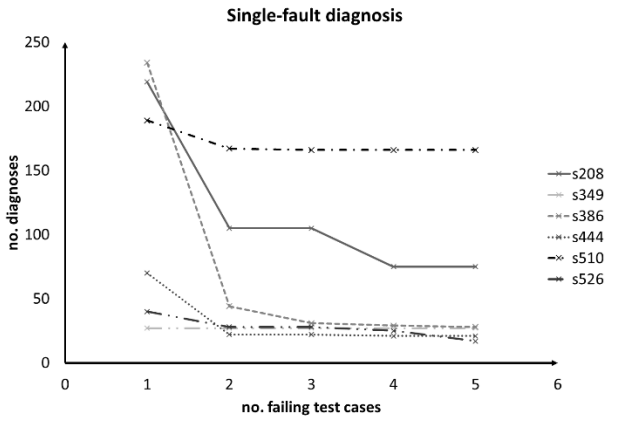


Figure 7: Single-fault diagnoses with increasing number of failing test cases.

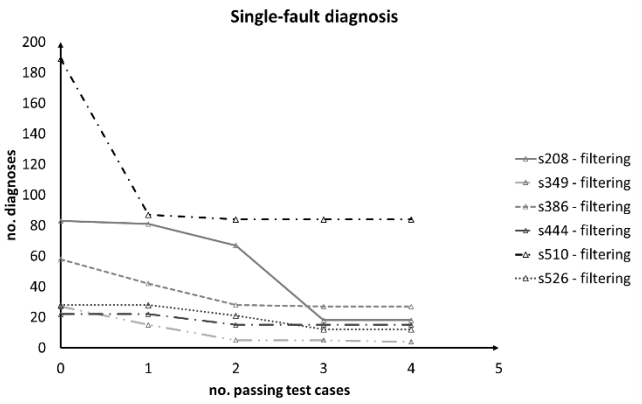


Figure 8: Improvements due to the filtering technique.

passing test cases. As in the previous experiment, the introduced fault always is among the retrieved set of single-fault diagnoses.

Regarding our experiments, passing test cases were able to further reduce the number of single-fault diagnoses for every program we considered.

Figure 9 outlines the running times for our algorithm we obtained for computing all single-fault diagnoses including the application of the filtering procedure. Remarkably, the random fault introduced in circuit s510 yields to a significant number of diagnoses and thus higher response times when compared to the remaining circuits. It appears that, (1) the structural complexity, (2) the random fault we introduced, and (3) the specific test cases revealing the introduced fault results in a (at least in relation to the other circuits) computationally expensive problem. On average we obtained 74 single-fault diagnoses corresponding to 44 faulty lines in the source code. Regarding our experiments, a designer can exclude over 93 percent of the statements and expressions from being faulty.

In a second series of experiments, we randomly injected two faults into every circuit we considered for our experiment.

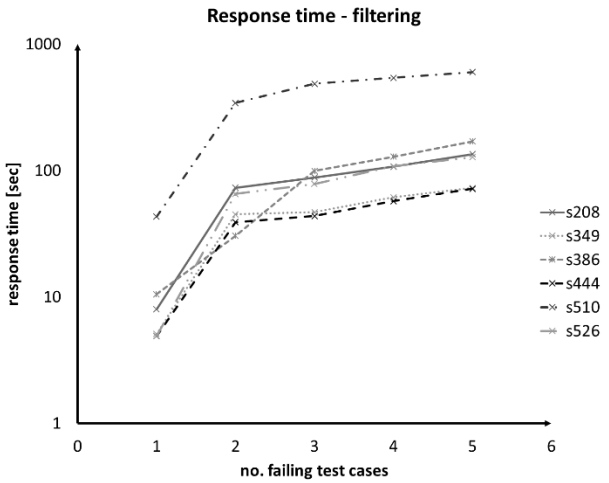


Figure 9: Running times for computing single-fault diagnoses.

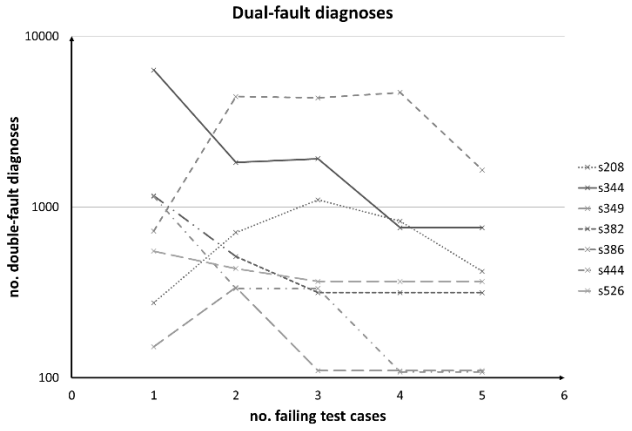


Figure 10: Dual-fault diagnoses for a part of the ISCAS'89 benchmarks

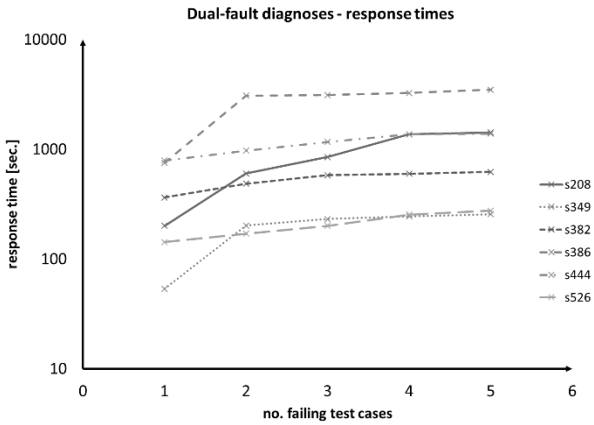


Figure 11: Running times for computing dual-fault diagnoses

Again, we generated up to five failing test cases and computed all single- and dual-fault diagnoses by using the algorithm introduced in Section IV. Figure 10 outlines the number of dual-fault diagnoses we obtained with increasing number of failing test cases.

For some circuits (e.g., s208, s349 and s386) the obtained number of dual-fault diagnoses is not monotonically decreasing. Unlike to computing single-fault diagnoses, this may happen due to the fact that test cases may mask some faults. For example, the first test case might reveal the first fault being introduced but may mask the second fault we introduced. As a consequence the second fault is not among the retrieved list of diagnoses. The second test case might reveal the second fault, therefore, after computing diagnoses, the second fault will also appear in the list of diagnoses. As a result, in the presence of multiple faults, when adding further test cases, in some cases, the number of diagnoses might increase. However, for most of the circuits in our experiment the number of dual-fault diagnoses decreases with increasing number of failing test cases. Again, for every circuit being considered in the experiment, the introduced pair of faults appeared among the computed dual-fault diagnoses

Figure 11 outlines the running times for computing dual-fault diagnoses. Note that one usually computes diagnoses in increasing order of cardinality. That is, we first use our algorithm to compute single-fault diagnoses and only if the real cause of misbehavior cannot be explained by a single-fault we continue with computing dual-fault diagnoses.

VII. Discussion and Related Work

The research work regarding fault localization is hardly comparable due to the variety of benchmarks and different approaches and abstraction levels being used. For this reason we predominately discuss the work where empirical results on the ISCAS'89 (and purely combinational ISCAS'85) benchmark suites have been obtained.

The authors of [29] propose a method that uses the crosstalk-induced pulse fault simulation to identify a set of suspected faults that are consistent with the observed responses. The authors propose two simulation-based methods to diagnose a crosstalk-induced pulse fault which may occur between the clock line of a flip-flop and a signal line in the sequential circuit. The first method is a basic method to diagnose the crosstalk-induced pulse faults. It uses information about the first and the last timeframe at which a crosstalk-induced pulse fault is detected. The second method uses additional information to reduce the computational complexity for diagnosing a crosstalk induced pulse fault. In order to reduce computational effort, the second method uses stored state information to calculate the primary output values at the present timeframe.

The authors of Boppana et al. [30] introduce a state information-based technique for supporting fault diagnosis. Storage of faulty state data corresponding can be used to reduce stored data. It has been demonstrated that the technique can support fast fault diagnosis in partial scan environments, particularly if the resulting design is acyclic. In doing so, the simplified structure of the partial scan circuit has been exploited. Arguably, the most important contribution of this work is the useful storage of faulty circuit data corresponding to flip-flops. Therefore, the circuit is no longer considered as a black-box element at diagnosis time and hence offers increased flexibility.

The authors of Smith et al. 2004 [31] present a SAT-based solution to design diagnosis of (also dual) faults where existing solvers can be utilized. The authors also examine different implementation trade-offs and heuristics. Like our work, experiments with dual faults demonstrate the efficiency and practicality of the approach.

Peischl and Wotawa 2006 [22] present work on VHDL and present results on the ISCAS'89 benchmarks regarding single test cases. This article introduces a model based on the MBD approach that abstracts over individual events within a single simulation cycle and allows one for performing source level debugging.

So far, the work on comparing the various debugging approaches is rather limited. In Finder et al. 2010 [32], a methodology is presented to evaluate debugging algorithms from a qualitative perspective. Notably, the authors of [32] lift the fault model originally defined for gate level net lists to higher level descriptions like HDLs and conclude that some

types of bugs can be handled using a simulation-based approach, while other types of bugs cannot be handled. Peischl and Wotawa [20] argue that simulation-based techniques may miss faults. Thus, the comparison of MBD-based techniques with simulation-based techniques can only be done in terms of case studies under a given set of assumptions (e.g., fault types) and the conclusions are restricted to the respective benchmarks being considered. Further, the different metrics to measure the quality or adequacy of the fault candidates (e.g., some notions of neighborhood, number of fault candidates, etc.) at the various levels of granularity (statements, expressions, operators etc.) make a generalized and meaningful comparison almost impossible.

The authors of [33] outline the results obtained with a fault-simulation based technique. The main differences to our experimental settings are as follows:

- In contrast to our experiments, the results described in [33] have been obtained on the gate level as this work does not focus on locating the erroneous statement or expression at the source code level.
- The fault localization is performed on optimized circuits rather than on the original design. The optimized circuits only comprise *AND* and *OR* gates. In contrast to that, our work deals with RTL designs and with locating the root cause on the source code level.
- The authors of [33] only specify the minimal length of the test cases and only give an upper limit of the number of failing and passing test cases.
- Most notably, and in contrast to our work, the technique introduced in [33] requires the primary input values and correct values for the primary output for every time frame. To our experience, a designer does only use limited correctness information (e.g., the expected signal values at the end of a test case) rather than having knowledge about all the intermediate values.
- The approach pursued in [33] fails for the circuits' s208 and s444 due to the high complexity for long failing test cases.
- Rather than allowing every statement, expression or signal to be faulty, only signals are considered as potential root cause for the observed misbehavior. This is a major difference to our approach, as our technique allows one for obtaining diagnosis candidates at the level of statements, and expressions including individual variables.
- Erroneous implementations are generated in terms of injecting a gate type error randomly after decomposition into *AND* and *OR* gates [33, 34].

Although the response times and the number of obtained diagnoses can hardly be compared due to the points mentioned above, our empirical results correspond with the results outlined in [33] in two respects:

- The number of potential single-fault diagnoses is reduced substantially when employing a couple of failing test cases.
- It appears that further increasing the number of failing test cases yields to saturation, i.e., the number of diagnosis candidates does not appear to become considerably smaller even when increasing the number of failing test cases substantially. In this respect, the decision to empirically investigate techniques that allow one for incorporating passing as well as failing test cases gains even more importance.

In the models used herein, we abstract over time (we use trace semantics) and variable values (we only operate with values *0* and *1*). This kind of abstraction is particularly suited for designs that can be synthesized. For mainstream programming languages (for example, the Java programming language) other abstractions like, for example, functional dependences or abstract interpretation models can be beneficial [35]. Finding a suitable abstraction is the key in successfully applying model-based software debugging, as this allows one for trading off computational complexity and accuracy of the obtained diagnoses. Today, it is an open issue, how to systematically find adequate abstractions and this issue requires further research.

Recent work approaches the fault localization problem merely from an algorithmic point of view. These articles differ from our research in two aspects. First, none of the works addresses source level debugging (in the sense of automatically highlighting the potential fault candidates at the level of expressions and statements at the HDL RTL level) and second, the evaluation of the novel algorithms and techniques is only performed on combinational circuits (mostly on the ISCAS'85 benchmark suite) and does not address sequential circuits (and thus the notion of state).

Siddiqi and Huang [36] propose a heuristic measurement point selection that can be computed efficiently. Furthermore, the technique introduced in [36] makes use of hierarchical diagnosis. For the largest system, where even this approach fails, the authors make use of specific abstractions. Unlike our approach (HDL trace semantics is an abstraction of the finer grained event semantics) this abstraction is not related to HDL language semantics. Experiments with the (combinational) ISCAS'85 benchmark suite indicate that this approach scales to all circuits in the suite except for one.

Feldman et al. [37] combine passive monitoring, probing and test sequencing with automated test pattern generation. Within their framework (FRACTAL), the authors empirically evaluate the trade-offs of three algorithms by performing experiments on the ISCAS'85 combinational benchmark circuits.

The same authors further propose a stochastic fault diagnosis algorithm called SAFARI [38], which trades off guarantees of computing minimal diagnoses for computational efficiency. In terms of the ISCAS'85 benchmarks, the authors empirically demonstrate that SAFARI achieves several orders of magnitude speedup over two well-known deterministic algorithms. The authors argue

that SAFARI can be of broad practical significance, as it can compute a significant fraction of minimal cardinality diagnoses for systems too large or too complex to be diagnosed by existing deterministic algorithms.

Abreu and van Gemund [39] use a heuristic approach to approximate the computation of minimal hitting sets and present a low-cost approximate minimal hitting set algorithm called STACCATO. The authors use a heuristic function that is particularly tailored to MBD problems. One difference to our work is that the candidates are not retrieved in increasing order of cardinality. Whether STACCATO can be tailored to software debugging (e.g., making use of STACCATO only for large problem spaces) is an open issue and subject of future research.

Like in this article, Bailey and Stuckey [40] present an incremental approach to compute hitting sets and show that circuit error diagnoses requires finding all minimal unsatisfiable subsets to compute minimal diagnoses. However, the proposed hitting set algorithm works in the context of constraints whereas the proposed algorithm herein is used in our automated debugging tool and is a variant of Reiter's hitting set algorithm dealing with sets of items.

VIII. CONCLUSION

In this article, we showed how to employ the well-founded theory of model-based diagnosis to fault localization in Verilog designs. We discussed today's simulation driven development lifecycle and proposed a model that can handle test suites comprising passing and failing test cases. To exploit passing test cases we used a technique called filtering and related this technique to Ackerman constraints. Regarding failing test cases, we used an iterative version of Reiter's hitting set algorithm.

We present an empirical evaluation of the impact of passing test cases alongside with an analysis of the running times of an iterative variant of Greiner et al.'s hitting set computation in the context of our automated debugging tool for HDLs. In this respect, we reported on exhaustive empirical results on one of the mostly employed benchmarks in the area of HDLs, the ISCAS'89 benchmark suite. Our results clearly indicated that exploiting test suites (comprising passing as well as failing test cases) considerably may improve the accuracy of the obtained diagnoses.

REFERENCES

[1] B. Peischl, N. Riaz, and F. Wotawa, "Using filtering to improve value-level debugging of verilog designs," In VALID 2013, The Fifth International Conference on Advances in System Testing and Validation Lifecycle, pp. 49–54, 2013.

[2] Z. Navabi, "VHDL: Analysis and Modeling of Digital Systems," McGraw-Hill, 1993.

[3] IEEE, IEEE Standard Verilog Language Reference Manual (LRM), IEEE STD 11364-1995, 1995.

[4] M. J. C. Gordon, "Relating event and trace semantics of hardware description languages," The Computer Journal, vol. 45, no. 1, pp. 27–36, 2002.

[5] S. Bailey, "Comparison of VHDL, Verilog and SystemVerilog," Digital Simulation White Paper, http://boydtechinc.com/btf/archive/att-1977/01-LanguageWhitePaper.pdf (last accessed on 09.05.2014).

[6] R. Abreu, P. Zoeteweij, and A. J C Van Gemund, "On the accuracy of spectrum-based fault localization," In Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007, TAICPART-MUTATION 2007, pp. 89–98, 2007.

[7] B. Baudry, F. Fleurey, and Y. Le Traon, "Improving test suites for efficient fault localization," In Leon J. Osterweil, H. Dieter Rombach, and Mary Lou Soffa, editors, ICSE, pp. 82–91, ACM, 2006.

[8] D. Hao, L. Zhang, T. Xie, Hong Mei, and J. Sun, "Interactive fault localization using test information," Journal of Computer Science and Technolgy, vol. 24, no. 5, pp. 962–974, 2009.

[9] B. Liblit, M. Naik, A. X Zheng, A. Aiken, and M. I. Jordan, "Scalable statistical bug isolation," ACM SIGPLAN Notices, vol. 40, no. 6, pp. 15–26, 2005.

[10] Y. Yu, J.A. Jones, and M.J. Harrold, "An empirical study of the effects of test-suite reduction on fault localization," In Software Engineering, 2008. ICSE '08, ACM/IEEE 30th International Conference on, pp. 201–210, 2008.

[11] B. Peischl, N. Riaz, and F. Wotawa, "Automated Debugging of Verilog Designs," International Journal of Software Engineering and Knowledge Engineering, vol. 22, no. 5, pp. 695–723, 2012.

[12] B. Peischl and F. Wotawa, "Model-Based Diagnosis or Reasoning from First Principles," IEEE Intelligent Systems, vol. 18, no. 3, pp. 32-37, IEEE Computer Society, May/June 2003.

[13] J. de Kleer, A. K. Mackworth, and R. Reiter, "Characterizing diagnoses," In AAAI , Howard E. Shrobe, Thomas G. Dietterich, and William R. Swartout, editors, pp. 324–330, AAAI Press / The MIT Press, 1990.

[14] R. Reiter, "A theory of diagnosis from first principles," Artificial Intelligence, vol. 32, no. 1, pp. 57-95, 1987.

[15] R. Greiner, B. A. Smith, and R. W. Wilkerson, "A correction to the algorithm in Reiter's theory of diagnosis," Artificial Intelligence, vol. 41, no. 1, pp. 79-88, 1989.

[16] S. H. Kan, Metrics and models in software quality engineering, Addison-Wesley, 1995.

[17] B. Peischl, V. R. Torrents, A. Kalchauer, S. Lang, "Business intelligence in software qualiy monitoring: Experiences and lessons learnt from an insutrial case study," In Proceedings of the 6th Software Quality Days (SWQD 2014), pp. 34-47, 2014.

[18] L. Lavazza and M. Mauri, "Software process measurement in the real world: Dealing with operating constraints," In Lecture Notes in Computer Science, Qing Wang, Dietmar Pfahl, DavidM. Raffo, and Paul Wernick, editors, Software Process Change, vol. 3966, pp. 80–87, Springer Berlin Heidelberg, 2006.

[19] B. Jobstmann, R. Bloem, A. Cimatti, G. Auerbach, and M. Moulinn, "Prosyd: Property-based system design, deliverable 2.1/1," PROSYD Techical Report, FP6-IST-507219, 2005.

[20] B. Peischl and F. Wotawa, "Error traces in model-based debugging of hardware description languages," In Proceedings of the Sixth International Symposium on Automated Analysis-driven Debugging, AADEBUG'05, pp. 43–48, New York, NY, USA, ACM, 2005.

[21] B. Peischl and F. Wotawa, "Computing diagnosis efficiently: A fast theorem prover for propositional horn theories," 14th Internaltional Workshop on Principles of Diagnosis (DX-03), pp. 175–180, June 2003.

[22] B. Peischl and F. Wotawa, "Automated source-level error localization in hardware designs," IEEE Design & Test of Computers, vol 23, no. 1, pp. 8-19, January 2006.

[23] F. Wotawa, "Applying Model-Based Diagnosis to SoftwareDebugging of Concurrent and Sequential ImperativeProgramming Languages," PhD thesis, Technische Universität Wien, 1996.

[24] O. Raiman, J. de Kleer, V. A. Saraswat, and M. Shirley, "Characterizing non-intermittent faults," In AAAI, Thomas L. Dean and Kathleen McKeown, editors, pp. 849–854, AAAI Press / The MIT Press, 1991.

[25] W. Ackermann, Solvable Cases of Decision Problems, North Holland, 1954.

[26] F. Wotawa, "Debugging hardware designs using a value-based Model," Applied Intelligence, vol. 16, no. 1, pp. 71-92, 2002.

[27] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," In IEEE International Symposium on Circuits and Systems, pp. 1929–1934, 1989.

[28] D. Nayak and D. M. H. Walker, "Simulation-based design error diagnosis and correction in combinational digital circuits," In VTS, pp. 70–79, IEEE Computer Society, 1999.

[29] H. Takahashi, M. Phadoongsidhi, Y. Higami, K.K. Saluja, and Y. Takamatsu, "Simulation-based diagnosis for crosstalk faults in sequential circuits," In Proceedings to the 10th Asian Test Symposium, pp. 63–68, 2001.

[30] V. Boppana, I. Hartanto, and W.K. Fuchs, "Fault diagnosis using state information," In Proceedings of Annual Symposium on Fault Tolerant Computing, pp. 96–103, 1996.

[31] A. Smith, A. Veneris, and A. Viglas, "Design diagnosis using boolean satisfiability," In Proceedings of the 2004 Asia and South Pacific Design Automation Conference, ASP-DAC '04, pp. 218–223, Piscataway, NJ, USA, 2004, IEEE Press.

[32] A. Finder and G. Fey, "Evaluating debugging algorithms from a qualitative perspective," In Forum on Specification Design Languages (FDL 2010), pp. 1–6, 2010.

[33] S.-Y. Huang, K.-T. Cheng, K.-C. Chen, and J.-Y. J. Lu, "Fault-simulation based design error diagnosis for sequential circuits," In Proceedings of the Design Automation Conference, 1998, pp. 632–637, 1998.

[34] A. Kuehlmann, D. I. Cheng, A. Srinivasan, and D. P. LaPotin, "Error diagnosis for transistor-level verification," In Proceedings of the 31st Annual Design Automation Conference, DAC '94, pp. 218–224, New York, NY, USA, 1994, ACM.

[35] W. Mayer and M. Stumptner, "Model-based debugging using multiple abstract models," In Fifth International Workshop on Automated and Algorithmic Debugging, CoRR, pp. 55-70, cs.SE/0309030, 2003.

[36] S. Siddiqi and J. Huang, "Sequential diagnosis by abstraction," Journal of Artificial Intelligence Research, vol. 41, no. 2, pp. 329–365, May 2011.

[37] A. Feldman, G. Provan, and A. van Gemund, "A model-based active testing approach to sequential diagnosis," Journal of Artificial Intelligence Research, vol. 39, no. 1, pp. 301–334, September 2010.

[38] A. Feldman, G. Provan, and A. Gemund, "Approximate model-based diagnosis using greedy stochastic search," In of Lecture Notes in Computer Science, Abstraction, Reformulation, and Approximation, Ian Miguel and Wheeler Ruml, editors, volume 4612, pp. 139–154, Springer Berlin Heidelberg, 2007.

[39] R. Abreu and A. J. C. van Gemund, "A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis," In SARA, Vadim Bulitko and J. Christopher Beck, editors, AAAI, pp. 2-9, 2009.

[40] J. Bailey and P. J. Stuckey, "Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization," In Proceedings of the 7th International Conference on Practical Aspects of Declarative Languages, PADL'05, pp. 174–186, Berlin, Heidelberg, 2005, Springer-Verlag.

Long-term Sustainable Knowledge Classification with Scientific Computing: The Multi-disciplinary View on Natural Sciences and Humanities

Claus-Peter Rückemann
Westfälische Wilhelms-Universität Münster (WWU),
Leibniz Universität Hannover,
North-German Supercomputing Alliance (HLRN), Germany
Email: ruckema@uni-muenster.de

***Abstract*—This paper presents the methodological and technical results of creating long-term sustainable knowledge resources, which can be used for documentation, classification, and structuring as well as with scientific discovery and deployment of supercomputing resources for advanced information systems. The focus is on the multi-disciplinary knowledge view on disciplines from natural sciences and humanities. The basic requirements resulting from the long-years' cases studies are long-term knowledge resources providing structure and universal classification features. The paper discusses the state-of-the-art implementation of information structures and object representations used with universal classification and computation algorithms for multi-disciplinary, dynamical knowledge discovery. The combination of universal knowledge resources and computational workflows based on High End Computing (HEC) resources and Universal Decimal Classification (UDC) have been successfully used for the goal of creating efficient long-term sustainable Integrated Information and Computing System components. The paper presents practical implementation examples from a range of disciplines with references to natural sciences and humanities, e.g., geosciences, astrophysics, and archaeology. The long-term results show that the overall sustainability principally depends on the methodological and systematical creation of content, structure, and classification with the knowledge resources.**

Keywords–Scientific Computing; Sustainability; Knowledge Resources; Multi-disciplinarity; Integrated Systems; Information Systems; Classification; UDC; Natural Sciences; Humanities.

I. INTRODUCTION

This paper presents the results and development of applications from long-term sustainable knowledge classification focussing on the multi-disciplinary aspects of natural sciences and humanities. The work and implementation are based on the creation of sustainable knowledge resources supporting structure, classification, and scientific supercomputing for any object and discipline [1]. Systematical and methodological developments are the major sources of contributions for long-term sustainable infrastructures. Technical developments complement the sustainability efforts, contributing to short- and medium-term success. With this context the amount of data as well as the complexity of information keeps steadily increasing. The developments of the last decades have shown that for a continuous positive progress not only the efficiency must increase, the more, developments must be made long-term sustainable, too. As the knowledge gathered during generations should be considered the most important component to the overall success we need universal knowledge resources that can handle documentation as well as universal classification and structuring. As being consent with most contributing disciplines and claimed by scientific councils, the knowledge resources should not only be traditional collections as with digital libraries [2] and isolated content [3] but, despite any challenges be accessible with scientific supercomputing resources in order to create advanced information systems and implement and improve workflows and recommended operation [4], [5].

The created features of the knowledge resources presented for the first time in this paper contain new practical concepts for information structures and object representations. The objects and derivatives, described in this paper, can be used with universal classification and computation algorithms for multi-disciplinary, dynamical knowledge discovery. This paper presents examples from archaeology and geosciences disciplines, resulting from practical case studies on structure and workflow modularisation, within the GEXI collaborations [6]. These are part of a multi-disciplinary knowledge structure. Further, the implementation of the knowledge objects is suitable to be used very flexibly with workflows on HEC resources, e.g., with IICS components [7], [8]. Multi-disciplinary knowledge resources are used to resemble and document of any information available. The requirements for complexity can become arbitrary high so that performant compute resources have to be used for any more advanced applications. The applicability for parallelisation of the contributing algorithms with the complex knowledge trees has therefore been analysed with the case studies. The motivation for investigating in the efficiency and modularisation of the knowledge trees is the increased potential for drastical improvements of the Quality of Data (QoD) with the result matrix, which contributes to advanced cognition within the multi-disciplinary context.

This paper is organised as follows. Sections II and III introduce with sustainability and vitality of knowledge-based architectures and main issues of complexity. Sections IV and V discuss the complexity and present a practically used classification approach to the challenges. Sections VI and VII describe the structure and challenges. Sections VIII and IX introduce the new concept of object carousels, the discovery of "missing links", workflows, references chains, and computational demands. Sections X and XI discuss the lessons learned and summarise conclusions and future work.

II. Previous work

Knowledge creation and knowledge management [9], [10] have been studied for more than twenty years now. Anyhow, so far long-term and sustainability issues have not really been considered in practice, especially in universal multi-disciplinary knowledge context. For example, there have been numberless approaches on knowledge management considering small isolated ranges of classical disciplines or defined purposes but not with multi-disciplinary approaches. Knowledge management (UDC:005.94 Knowledge management) is obvious to be only one of the many aspects of knowledge (UDC:0 Science and knowledge), from creation to organisation and universal and long-term sustainable development and use [11].

For all components presented in this paper, the main information, data, and algorithms are provided by the LX Foundation Scientific Resources [12], e.g., the volcanological data, the meteorite crater data, and archaeological data.

Information about the following data sources has been integrated and deployed with the knowledge resources for the previous basic case studies and developments. So, the creation of long-term knowledge resources decisively contributes to the goal of a successful creation of long-term sustainable Integrated Information and Computing System (IICS) components.

The referred "Leibniz" data (see the following references on Gottfried Wilhelm Leibniz, 1646–1716) has been included into the workflow chains, e.g., creating historical associations with the the content of archaeology and geosciences will otherwise not be accessible. An example is the communication regarding volcanoes, earthquakes, and caves in manuscripts and letters or content of pictorial realia objects, which are not available via search engines. From the Leibniz sources there is a rich contribution for the result matrix on volcanism, volcanology, and geology by various historical objects, references, and sources, especially for volcanism, Vesuvius [13], as well as earthquake related context [14], even from concept glossaries [15], manuscript collections and catalogues [16], [17] as, e.g., [18], [19], or Leibniz related copperplates [20]. For example, the "praehistoric unicorn" reconstruction [21], as well as material on geological context has not been referenced before from knowledge resources' objects and is not freely and publicly available as a direct reference, media or verification [22].

Material in specialised collections, for example in the European Cultural Heritage Online [23] would not be easily accessible due to the type and context of the material.

Further data being publicly available can be incorporated in any way under the premise that the data formats are accessible and interfaces have been provided. An example is the CLImatological database for the World's OCeans (CLIWOC) [24], a climatological database for the world's oceans from 1750–1850, containing digitized data from logbooks of pre 1854 voyages of English, Spanish, Dutch, and French ships.

III. Sustainability and vitality

In the context of this research, the goal for "long-term" means > 50 years. The long-term strategy has been discussed in detail with previous implementations [25]. Data mining is not only an analysis step of knowledge discovery in databases based on informatics but much more general in data pools. It is an inter-disciplinary as well as multi-disciplinary field of many sciences and computer science. It means discovering patterns in data pools using methods implementing statistics, classification, artificial intelligence, learning and many more based on knowledge resources. The process targets to extract information from knowledge resources and gaining content and context, e.g., based on structure and references, in order to prepare for further use. Sustainable long-term strategies have to combine operation, services, and especially the knowledge resources [26], [1]. With the available systems components, we have Resources Oriented Architectures (ROA), Services Oriented Architectures (SOA), and "Knowledge Oriented Architectures" (KOA) in addition [27]. For long-term operation, all three must be obtained from the creation and operation. Considering the entirety of aspects necessary for a successful long-term change management with future information technology structures. Nevertheless, the KOA is the most important complement as it contains the highest percentage of the overall investments for the results and the data that may even not be reproducible later on.

IV. Complex knowledge resources case

Central aspects for uses cases are the definition of knowledge and the features of the knowledge resources.

A. Knowledge definition

In general, we can have an understanding, where knowledge is: Knowledge is created from a subjective combination of different attainments as there are intuition, experience, information, education, decision, power of persuasion and so on, which are selected, compared and balanced against each other, which are transformed and interpreted.

The consequences are: Authentic knowledge therefore does not exist, it always has to be enlived again. Knowledge must not be confused with information or data which can be stored. Knowledge cannot be stored nor can it simply exist, neither in the Internet, nor in computers, databases, programs or books. Therefore, the demands for knowledge resources in support of the knowledge creation process are complex and multifold.

There is no universal definition of the term "knowledge", but UDC provides a good overview of the possible facets. For this research the classification references of UDC:0 (Science and knowledge) define the view on universal knowledge.

B. Knowledge resources

The knowledge resources created can integrate any object. These objects can be described with universal classification, handled with phonetic algorithms [28], [29], and can refer to external resources. The structure of the knowledge objects has to support the modularisation for application scenarios where the workflow has to allow highly efficient implementations itself. Creating workflows based on the multi-disciplinary knowledge matrices therefore requires highly performant resources. The overall big data challenges, data intensive volume,

variability, velocity and for future scenarios especially data vitality, meaning long-term documentation, usability, and accessibility can be handled in a scalable, modular way. Further, the components created are considered to become objects of sustainable knowledge resources, for long-term persistent big data vitality of documentation, processing, analysis, and evaluation. The created solution for long-term use meets a number of attributes, e.g., it should be generic, superior, adaptable, flexible, seminal sustainable. In summary, these combined vital features are called "eonic".

V. KNOWLEDGE RESOURCES CLASSIFICATION SUPPORT

The operated knowledge resources, based on the LX Foundation Scientific Resources [28], incorporate UDC classification support for any discipline and purpose, e.g., for knowledge discovery and workflows. Practical summarising excerpt subsets for specific disciplines used with the case studies presented here are given in Tables I, II, III, and IV. These subsets are use with the knowledge resources on classification regarding archaeology, volcanoes, impact events, and sinkholes.

Table I. ARCHAEOLOGY KNOWLEDGE RESOURCES CLASSIFICATION.

UDC Code	*Description*
UDC:902	Archaeology
UDC:903	Prehistory. Prehistoric remains, artefacts, antiquities
UDC:904	Cultural remains of historical times
UDC:930.85	History of civilization. Cultural history
UDC:"63"	Archaeological, prehistoric, protohistoric periods, ages
UDC:(23)	Above sea level. Surface relief. Above ground generally.
UDC:(24)	Below sea level. Underground. Subterranean
UDC:=14	Greek (Hellenic)
UDC:56	Palaeontology
UDC:55	Earth Sciences. Geological sciences
UDC:711.42	Kinds of town, locality, settlement
UDC:720.2	Architecture techniques and methods
UDC:720.31	Prehistoric architecture
UDC:720.32	Ancient architecture

Table II. VOLCANOES KNOWLEDGE RESOURCES CLASSIFICATION.

UDC Code	*Description*
UDC:532	Fluid mechanics in general.
UDC:550.8	Applied geology and geophysics. . . .
UDC:550.93	Geochronology. Geological dating. . . .
UDC:551	General geology. Meteorology.
UDC:551.1	General structure of the Earth
UDC:551.2	Internal geodynamics (endogenous processes)
UDC:551.21	Vulcanicity. Vulcanism. Volcanoes. Eruptive phenomena.
UDC:551.23	Fumaroles. Solfataras. Geysers. Hot springs. Mofettes.
UDC:551.24	Geotectonics
UDC:551.26	Structural-formative zones and geological formations
UDC:551.4	Geomorphology. Study of the Earth's physical forms
UDC:551.44	Speleology. Caves. Fissures. Underground waters
UDC:551.462	Submarine topography. Sea-floor features
UDC:551.5	Meteorology
UDC:551.588	Influence of environment on climate
UDC:551.7	Historical geology. Stratigraphy
UDC:551.8	Palaeogeography
UDC:552.2	General petrography. Classification of rocks
UDC:552.6	Meteorites
UDC:631	Agriculture in general
UDC:631.4	Soil science. Pedology. Soil research

Tables III and IV show the excerpts used for basic impact events and sinkholes classification.

Table III. IMPACT EVENTS KNOWLEDGE RESOURCES CLASSIFICATION.

UDC Code	*Description*
UDC:5	Mathematics and natural sciences
UDC:500	Natural sciences
UDC:539	Physical nature of matter
UDC:539.63	Impact effects
UDC:539.8	Other physico-mechanical effects

Table IV. SINKHOLES KNOWLEDGE RESOURCES CLASSIFICATION.

UDC Code	*Description*
UDC:519.2	Probability. Mathematical Statistics
UDC:556.34	Groundwater flow. Well hydraulics
UDC:624	Civil and structural engineering
UDC:624.151	Foundations. Foundation bed
UDC:699	Protection of and in buildings. Emergency measures
UDC:930.85	History of civilization. Cultural history
UDC:528.9	Cartography. Mapping (textual documents)
UDC:726.6	Cathedrals. Basilicas. Domes

The small unsorted excerpts of the knowledge resources objects only refer to main UDC-based classes, which for this part of the publication are taken from the Multilingual Universal Decimal Classification Summary (UDCC Publication No. 088) [30] released by the UDC Consortium under the Creative Commons Attribution Share Alike 3.0 license [31] (first release 2009, subsequent update 2012).

As one of the elementary qualities, the LX Foundation Knowledge Resources allow to refer to any kind of references, therefore they also allow to refer to any kind of classification. If nothing special is mentioned then all the basic classification codes are used in an unaltered way. If any classification refers to a modified code then the authors of contributions have to notice and document the modifications explicitely. The classification sets have been referred to and used with the presented computation. UDC [32] currently provides around 70,000 entries in about 100 top classes, whereas the UDC Summary [33] provides a selection of more than 2,000 classes. The multi-lingual support lists translations in about fifty languages [34], [35]. UDC classifications have been integrated with tens of thousands of knowledge objects [29], which are a base for each computation [34], [36], [11].

VI. STRUCTURE RELATED SUPPORT

For the components shown here, the structuring capabilities of the the LX Foundation Scientific Resources [12] have been deployed. Mostly any external information and types of objects can be integrated into this structure. The figures, object entries, and photo media samples shown in the passages of the following case study examples are computed from the content of the LX Foundation Scientific Resources. Figures, object entries, and photo media samples © C.-P. Rückemann, 2011, 2012, 2013, 2014).

Structuring information requires a hierarchical, multi-lingual and already widely established classification implementing

faceted analysis with enumerative scheme features, allowing to build new classes by using relations and grouping. This is synonym to the Universal Decimal Classification (UDC) [37]. In multi-disciplinary object context a faceted classification does provide advantages over enumerative concepts. Composition/decomposition and search strategies do benefit from faceted analysis. It is comprehensive, and flexible extendable. A classification like UDC is necessarily complex but it has proved to be the only means being able to cope with classifying and referring to any kind of object.

Copies of referred objects can be conserved and it enables searchable relations, e.g., for comparable items regarding special object item tags. The UDC enables to use references like for object sources, may these be metadata, media, BIBTEX sources or Digital Object Identifier (DOI) [38] as well as for static sources. With interactive and dynamical use for interdisciplinary research the referenced objects must be made practically available in an generally accessible, reliable, and persistant way. A DOI-like service with appropriate infrastructure for real life object services, certification, policies and standards in Quality of Service, for reliable long-term availability object, persistency policies should be available.

Therefore, for any complex application, these services must be free of costs for application users It would not be sufficient to build knowledge machines based only on time-limited contracts with participating institutions. These requirements include the infrastructure and operation so data availability for this long-term purpose must not be depending on support from data centers providing the physical data as a "single point of storage". Unstructured information, the data variety, is one major complexity. For relational databases, a lot of players providing offerings in this space go through the cycle of what the needs are for structured data. As one can imagine, a lot of that work is also starting for unstructured or semi-structured data with Integrated Systems. Data access and transfer for structured data, unstructured data, and semi structured data may be different and may to a certain extend need different solutions for being effective and economic [39].

The long-term objects must be able to contain the essential knowledge, even as medium- and short-term objects cannot be preserved or made persistent as, e.g., DOI (Digital Object Identifier), URN (Uniform Resource Name), URL (Uniform Resource Locator), and PURL (Persistent Uniform Resource Locator) will vanish and context and sources may fade away as well as OS (Operating System) features used. Therefore, we have to distinct between the real instance of a DOI and URL or a context situation and a descriptive reference of these objects. These descriptive references can contain as much information and knowledge as possible (for example DOI, URL, context description, sources).

VII. COPING WITH THE CHALLENGES

A. Modular components for geoscientific applications

Complex geophysical exploration is an explicit big data problem. Data locality and data movements are of essential importance. Therefore, data handling does take longer a time than the compute intervals. Due to the short intervals for licensing and the high costs even the time efficiency has to be increased. This can be supported by parallel techniques [40], [41], [42]. In many multi-disciplinary cases, e.g., explicitely shown with the case studies [7], [28], the more with growing importance of evaluation processes, the task- and thread-parallelity has to be increased both. The data in geosciences and in associated natural sciences contains the most valuable information because many of these natural processes change in geological time intervals. Imaging for oil and gas is one of the most demanding tasks in computational sciences. It requires scale-out architectures, the processing and simulation are computation intensive as well as data intensive. The data provides long-term challenges on knowledge and resources to researchers and industry because of expenses on data collection and long-term usability.

B. Rising requirements on quantity and computation

As soon as even a selected subset of the available classification is integrated with a subset of detailed knowledge resources, the requirements for computing and interfaces are rising drastically. The increasing demands for advanced scientific computing are resulting from the huge number of relations within the knowledge resources as well as a consequence of the workflows, dynamical interaction, presentation, and visualisation of results. The conditions for the optimal computing architecture are defined by the application scenario, not by the knowledge resources themselves.

C. Quality for Quantity

For the discovery of a result matrix from a large quantity of data, additional high quality resources can be used for improving the quality of results deduced. The premise is that appropriate workflows and algorithms will be applied. The high quality knowledge resources have been used as "Quality for Quantity" (Q4Q), in order to build any additional missing references in the quantity data. With these HEC and discovery processes, big data means volume regarding storage, means variability regarding workflow processes, means velocity regarding instances, and vitality regarding knowledge resources.

VIII. OBJECT CAROUSELS

The organisation of the knowledge objects can be arbitrary complex. Many cases can be described in a simplified way like a mindmap, which has been used for introducing a new symbolic representation named "object carousels". The knowledge objects build a kind of dynamical molecules. These molecules have connectors and references. These connectors can connect with other knowledge objects by computing references from any number of directions. The process reminds of rotating branches of trees, rings, and multi-dimensional objects for finding pluggable connections. The creation of object carousels does have the benefit, that knowledge discovery workflows can be implemented very scalable, using various algorithms for connecting trees. For example, full text references can be used between any carousels in order to compute a result matrix.

A. Object mapping

The mapping in Figure 1 shows an excerpt for the volcanology context on terrestrial volcanism calculated from the knowledge resources. These allow to calculate relations via flexible, user-defined algorithms.

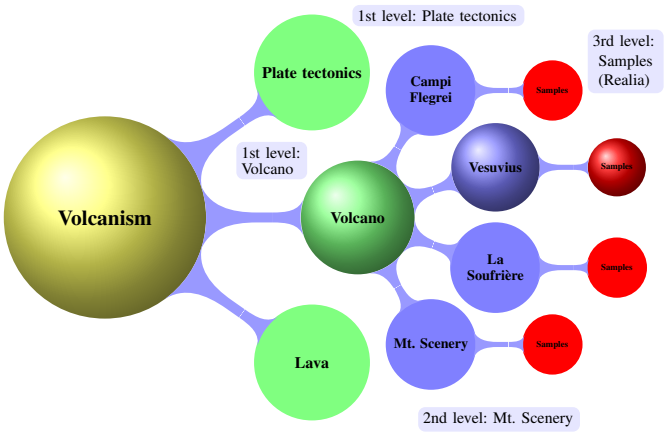


Figure 1. Object carousel for terrestrial "volcanism" context with subset of computed volcano references and examples of levels object relations.

The figure shows an excerpt of the direct relations by quality of relations and quality of objects. The colours visualise object groups or attributes within each figure. Any object or attribute can dock-in at any placed defined by the workflows, not depending on the grouping. Nevertheless, the decision within the workflow maybe assisted by the group information. The knowledge resources can contain objects and relations as well as classification entries. The case study example being the base for illustrating the different aspects in all the next sections follows a discovery path (3D), starting discovery on object and realia references in the volcanology dimension.

B. Information and object usage

In a non-promoted environment, a knowledge search engine showed significant requirements with up to over 500,000 application- and several million object-requests per day. The study on object usage from international public interest groups done in a time interval from 1994 to 2012 [6] revealed comparable large numbers of accesses and complexity. The object mapping is a basic part, whereas the algorithmic workflow for improving the quality can be as expendable as using every information available with each step recursively and iteratively. The computation share can increase to hours per discovery instance but computation can be done for any number of carousels in parallel. The KOA opens flexible support for task and process parallelity for using objects and object groups or clusters.

C. Case study views

Suitable views for volcanoes are: Type (of volcano, coarse categories), date on timeline, size (height). For craters respective views are: Type (of crater, fragmentary), date on timeline, size (diameter). An object carousel generated for volcano types, shows the knowledge resources groups (Figure 2).

Figure 2. Object carousel for volcano and type references computed for terrestrial volcanism, providing volcano type references.

An object carousel generated for impact craters, shows the different types present in the knowledge resources groups and their crater categories (Figure 3).

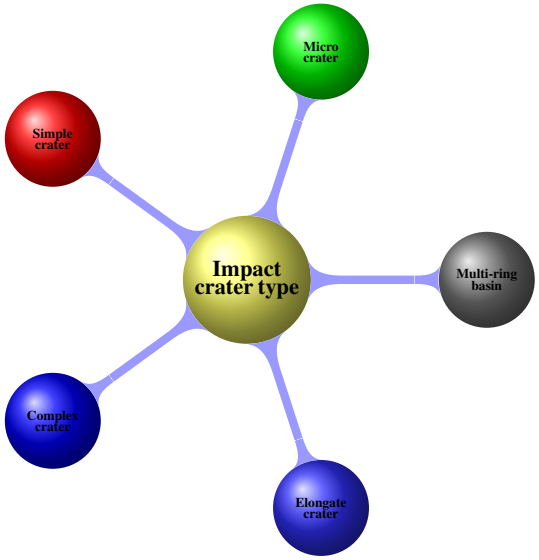


Figure 3. Object carousel computed for impact crater categories.

Criteria for impact crater classification are:

- Size of the impacting object,
- Speed of the impacting object,
- Material of the impacting object,

- Composition and structure of the target rock,
- Angle that the impacting object hits the target,
- Gravity of the target object respective planet,
- Porosity and other attributes of the impacting object,
- Age of the impact,
- Size of the impact,
- Structure of the crater.

Impact crater indicators, for example:

- Planar fractures in quartz,
- Shocked quartz,
- Glass fragments.

If approaching from the "catastrophe" view it has shown that the most prominent relation is the "size". This mostly correlates with "diameter" and still mapping and timelining will come natural.

A comparable object carousel for impact crater types and geological period references is provided in Figure 4.

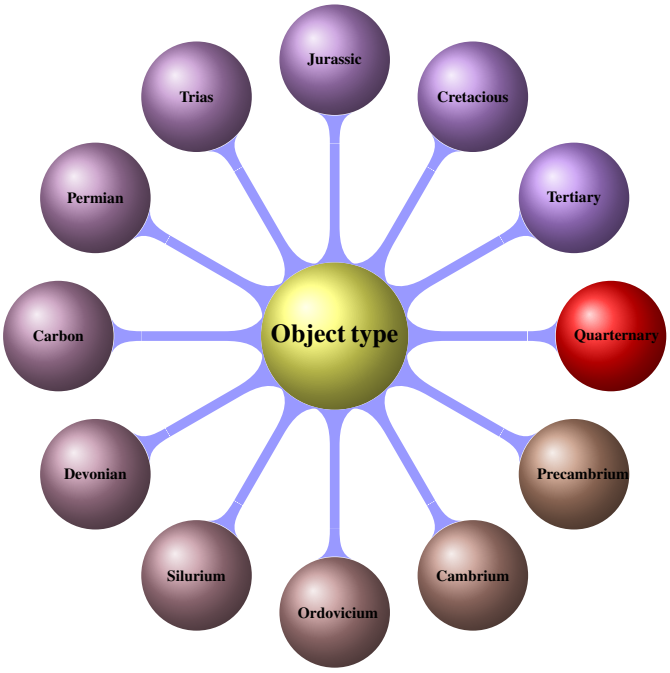


Figure 4. Object carousel object type for computed period references (terrestrial volcanoes, impact craters, and geological processes).

An evaluation of the association that users have, showed that the criteria "date" and "location" are most prominent with objects if the workflow approaches from the "surface (of the earth)" view. Therefore, mapping and timelining will be the natural result.

D. Improving quality within the workflow

The resources, workflow, and classification are essential for a high level of usability quality of results. The elaborate workflow process for improving the quality of results when calculating a result matrix from a knowledge base is:

a) Calculate associator attributes and classes,
b) Compute on a base with large numbers of objects,
c) Evaluate detailed classification information,
d) Compute for a reduction of numbers of objects,
e) Create suggestions and recommendations.

The first computation block (b) is needed for considering more objects when applying the further steps afterwards. Here, the classification is essential for improving the quality for the respective selection process. The second computation block (d) is necessary for improving a selection process for the target audience or services. The selection processes can be significantly supported by high quality knowledge resources (Q4Q), e.g., via the authored, classified, and audited content, with regular expression search, and phonetical algorithms.

E. Improving coverage: Dark data

In analogy to "dark matter" and "dark energy", there exists "dark information" and to an uncertain extent an unknown driving force in knowledge creation, even building "dark service" provided via "dark resources". Those information resources are not wider accessible and it is not known where the intention of gathering and creation. Anyway, this information must be considered for any holistic long-term concept as it provides an important factor for the overall knowledge and will stay in existance despite of any development. With the concept of long-term knowledge resources the information has been integrated in order to extend the base for any knowledge discovery. Considered methods for integration of resources are, e.g., references, description or caches. This further includes seamless updating of information, licensing of resources, dynamical use of data as well as provisioning of defined quality and reliability for sources and complements.

IX. Discovery of Missing Links

From the disciplines of humanities and archaeology, the directed tree spanning from settlements to used materials will show up with a practically defined depth. On the other hand, starting from natural sciences a directed tree spanning to materials associated with processes will deliver a natural sciences path. Along with the different paths, the genetic connectors of both carousels will show up with links from both directions. The connecting links, or short "connections" from the directed search do open new associations that can be used to discover the overall knowledge much deeper with new facets and quality, which provide multi-disciplinary links that have been missing in non-genetic discovery.

In general, any kind of tree path can be generated from the knowledge resources using a workflow and any number of carousels can be discovered for connections. The following example shows a simple two-carousel case (Figure 5). Computing the object carousels connections is shown for a historical city carousel and an environment object carousel. The trees show a subset of computed references computed by the workflow within the knowledge resources. The depth of the trees may by different for the computation. The connections are considered as soon as they lead to a defined conformity. In that case, defined conformity can mean comparable or identical. The example shows two trees, one from archaeology and one

from natural sciences disciplines. For both, at a certain branch leading to object referring to stone material, which is shown by the highlighted red bullets.

A. Computing connections on modular objects

Figure 5 shows the principle used for computing connections with object carousels. It depicts one fitting branch, within archaeology and geosciences associated objects. Starting with the objects `HistoricalCity` and `Environment` (identified by large golden bullets) and the linking objects "stone" the computed carousels show trees with a subset of references. The workflow attributes have been choosen to provide no tree depth restriction for computation. The two fitting connection lines within the object carousels of this example are highlighted in a three-dimensional representation: `Roman:Pompeji:Napoli:Architecture:Volcanicstone` and `Volcanology:Catastrophe::Volcanicstone`. In the sample workflow, the carousel connections are calculated via non-explicit references of comparable objects (red objects) from knowledge resources within trees. In addition, the red circle does mark those objects at the same depth level, including the fitting object term for historical city and environment `Volcanicstone`. The excerpt of associated multi-disciplinary branch level objects are `Limestone`, `Impactfeature`, and `Climate change`.The method for creation of non-explicit references can be defined in the workflow. Here, full text mining and evaluation (red objects) has been used. For derivated associations additional objects can be computed and extracted in every branch as well as on all levels.

B. Connecting knowledge

Objects can be connected by various attributes. These may be attributes associated with content as well as with context. For example, relations for a volcano object can be connected and triggered by a large variety of attributes. Table V shows an excerpt of attributes and examples.

Table V. ATTRIBUTES LINKING AND TRIGGERING VOLCANO OBJECTS AND SELECTED EXAMPLES (EXCERPT).

Attribute	*Example in Archaeology/Geosciences*
`Time`	Events on timeline
`Location`	Volcano-impact-settlement locations
`Physics`	Earthquakes
`Chemistry`	Volcanic SO_2 ejection
`Geology`	Earth crust, petrography
`Catastrophes`	Volcanic eruptions, Tsunamis
`Etymology`	Phlegra, Vesuvius
`Cults, religions`	Volcano gods
`Artefacts`	Archaeological objects, "Pompeji" events
`Historic events`	Volcano, climate, economy, revolution

Relations can refer to any multi-disciplinary topic, building results from combination of information and generation of new objects and references, e.g., visualisations and views.

Selecting "catastrophe" categorised objects from the knowledge resources results in a matrix including groups of keywords, for example:

1) `Meteorite, impact, Yucatan, Mexico, dinosaurs, extinct, Cretaceous, CT boundary, catastrophe.`
2) `Volcano, eruption, Santorini, Thera, crete, Minoan civilisation, culture, Mycenae, culture, fleet, volcanic ash, vanish, rise, historical city, catastrophe.`
3) `Volcano, Vesuvius, Campi Flegrei, phlegra, scene of fire, Pompeji, Herculaneum, volcanic ash, lapilli, catastrophe.`
4) `Solfatara, volcano, Vesuvius, Campi Flegrei, phlegra, scene of fire, Pompeji, Herculaneum, volcanic ash, lapilli, catastrophe.`

For example, following an etymological tree leads from 'Vesuvius — Campi Flegrei' to "phlegra" greek for 'scene of fire'. The above keyword groups resolve to object entries, for example

1) `Chicxulub`,
2) `Thera, Santorini`,
3) `Vesuvius, Pompeji`,
4) `Solfatara`.

. These objects refer to media samples as shown with some examples for Vesuvius, Pompeji, and Solfatara.

The following excerpt contains some structure, UDC classification, keywords, references, and satellite image reference. The references for the geopositioning are created via classification and can be used for any purpose. Listing 1 shows an excerpt of an LX Resources object entry [43], "Vesuvius" volcano.

```
1  Vesuvius [Volcanology, Geology, Archaeology]:
2           (lat.) Mons Vesuvius.
3           (ital.) Vesuvio.
4           (deutsch.) Vesuv.
5           Volcano, Gulf of Naples, Italy.
6           Complex volcano (compound volcano).
7           Stratovolcano, large cone (Gran Cono).
8           Volcano Type: Somma volcano,
9           VNUM: 0101-02=,
10          Summit Elevation: 1281\UD{m}.
11          The volcanic activity in the region is observed
             by the Oservatorio
12          Vesuviano. The Vesuvius area has been declared a
              national park on
13          \isodate{1995}{06}{05}. The most known antique
             settlements at the
14          Vesuvius are Pompeji and Herculaneum.
15          Syn.: Vesaevus, Vesevus, Vesbius, Vesvius
16          s. volcano, super volcano, compound volcano
17          s. also Pompeji, Herculaneum, seismology
18          compare La Soufrière, Mt. Scenery, Soufriere
19          %%IML: UDC
             :[911.2+55]:[57+930.85]:[902]"63"(4+23+24)
             =12=14
20          %%IML: GoogleMapsLocation: http://maps.google.de
             /maps?hl=de&gl=de&vpsrc=0&ie=UTF8&ll
             =40.821961,14.428868&spn=0.018804,0.028238&t=h&
             z=15
```

Listing 1. Knowledge resources – object entry "Vesuvius" volcano.

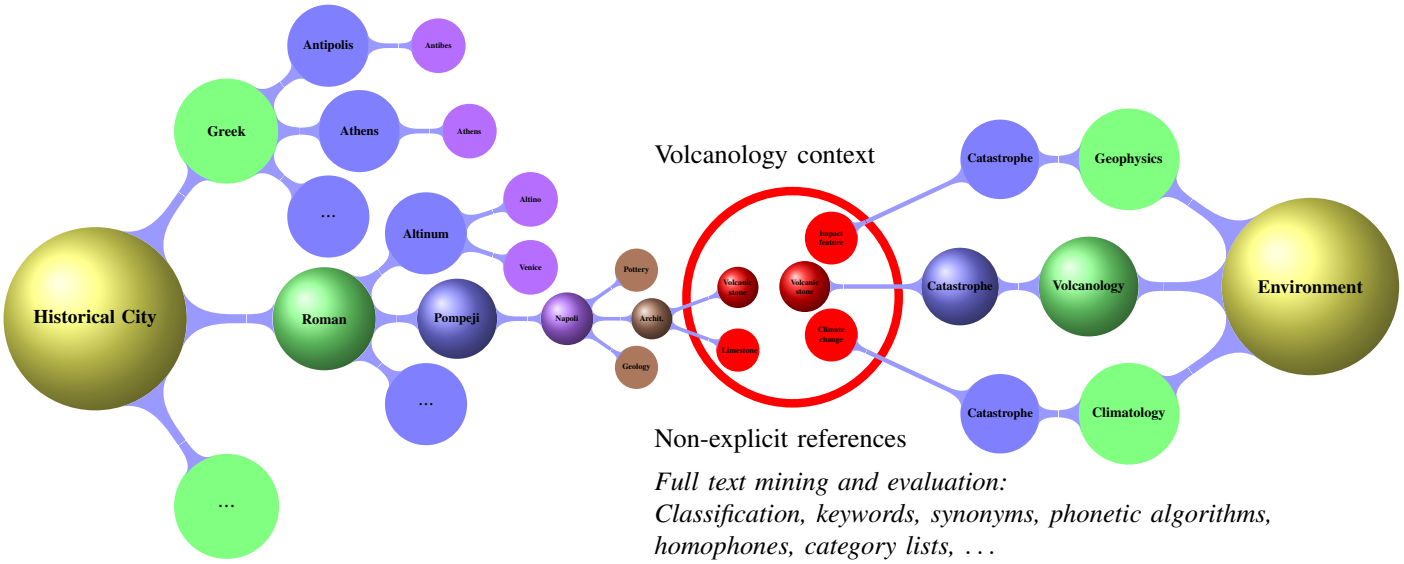


Figure 5. Computing object carousel connections: Historical city and environment object carousels showing trees with a subset of computed references. In this sample workflow the carousel links are calculated via non-explicit references of comparable objects (red) from knowledge resources within trees.

The example contains a reference and VNUM for the Vesuvius volcano, various secondary objects, UDC classification, satellite image reference and, e.g., refers to “Soufriere”, “La Soufrière”, and “Mt. Scenery”.

Listing 2 lists an entry excerpt for realia material associated with the Vesuvius volcano.

```
1  Object:              Volcanic material.
2  Object-Type:         Realia object.
3  Object-Location:     Vesuvius, Italy.
4  Object-FindDate:     2013-10-00
5  Object-Discoverer:   Birgit Gersbeck-Schierholz, Hannover,
     Germany.
6  Object-Photo:        Claus-Peter Rückemann, Minden,
   Germany.
7  Object-Relocation:   Claus-Peter Rückemann, Minden,
   Germany.
8  %%IML: media: ... img_2402.jpg
9  %%IML: media: ... img_3823.jpg
10 %%IML: media: ... img_3824.jpg
11 %%IML: UDC-Object:[551.21+55]:[911.2](37+4+23)=12
12 %%IML: UDC-Relocation:069.51+(430)+(23)
13 %%IML: cite: YES 20130000 {LXK:High End Computing;
   Knowledge resources; Objects; Archaeology; Geosciences;
   Vesuvius; Pompeji} {UDC:...} {PAGE:----..----} LXCITE://
   Rueckemann:2013:Computing_Objects
14 %%IML: cite: YES 20140000 {LXK:Nature; History; Napoli;
   Vesuvius; Pompeji} {UDC:...} {PAGE:----..----} LXCITE://
   Gersbeck:2014:Vesuvius
```

Listing 2. Knowledge resources – Entries for Vesuvius material.

Besides the UDC object and relocation data the excerpt carries the media references and citations and originating sources, researchers, and relocation for the realia objects.

Figure 6 illustrates the computed objects (Topicview), here the latest available volcanic samples for Vesuvius, after processing showing the variety of material from the Vesuvius volcano.

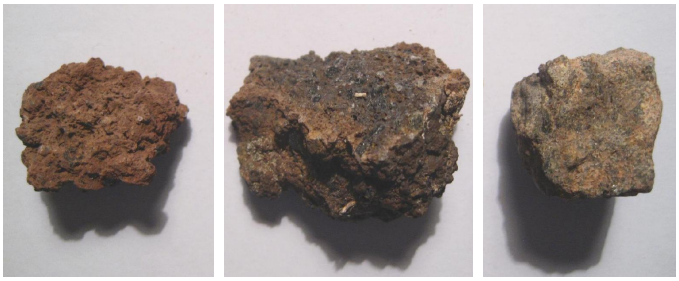


Figure 6. Topicview result matrix – Vesuvius realia objects (excerpt): Range of volcanic ashes and lapilli (Vesuvius, 2013).

Any of these objects being part of the resulting matrix for a request, e.g., photos for object entries as well as media data for physically available samples, have been found via references and UDC from the knowledge base (`UDC:551.21...`). The realia references for the objects refer to a collection where the samples are stored. Further analysis for the samples is available via the knowledge resources.

Figure 7 illustrates the computed objects (Topicview) of realia objects associated with “Vesuvius”.

Figure 7. Topicview result matrix – “Vesuvius” associated realia objects (excerpt, from left to right): Solfatara sample (2013), Pompeji lapilli (2013), Pompeji plaster sample (2013).

The selection criteria are “archaeology, artefacts, cultural remains of historical times, architecture techniques and methods, ancient architecture” (UDC:902, UDC:903.2, UDC:904, UDC:720.2, UDC:720.32). The associated views show a sample from the Solfatara (sample front side and sample rear side) near Vesuvius and a sample of a Pompeji plaster (Pompeji style I, incrustation style).

Figure 8 illustrates a specific selection (Topicview) of “Vesuvius” realia objects.

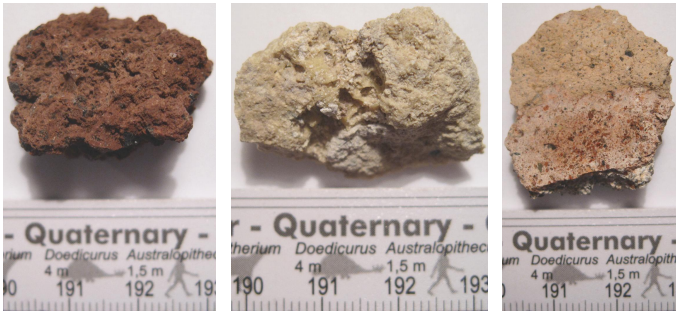


Figure 8. Topicview result matrix – “Vesuvius” and associated realia objects with measure (excerpt, from left to right): Vesuvius lapilli (2013), Solfatara sample (2013), Pompeji plaster sample (2013).

The associated views show a sample from the Solfatara (sample front side and sample rear side) near Vesuvius and a sample of a Pompeji plaster (Pompeji style I, incrustation style). The media samples shown are part of comprehensive classified object entry descriptions and the citations refer to [44]. Respective object entries have been shown and discussed in detail in the context of the component implementations [25].

C. Generation and combination of knowledge

The following visualisations paradigmatically illustrate results from the compute requests based on the content of the implemented content. An on-location attribute has been choosen for the relations in order to compute a distribution map for volcanological features using the lxlocation workflow. The location attribute is suitable for referring to an unlimited number of further multi-disciplinary information in this case. A sample distribution of classified terrestrial volcanological features is depicted in Figure 9.

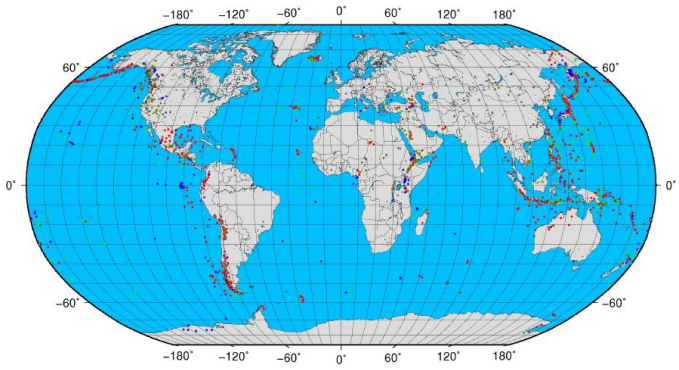


Figure 9. Volcanomap – worldmap for classified volcanological features.

The map is computed from the related object context contained in a volcanological features research database of the knowledge resources. The volcanological features are classified and several classification groups have been choosen for the result. The map shows the present situation according to the available volcanological data. The associated sample distribution of terrestrial impact features (meteorite) is depicted in Figure 10.

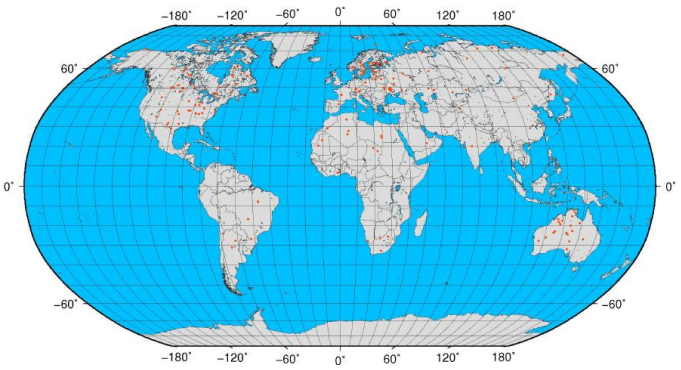


Figure 10. Impactmap – worldmap for impact features (meteorite).

The map is computed from the related object context (lxlocation workflow) contained in a meteorite impacts features research database of the knowledge resources.

The visualisations are based on the results gathered in the knowledge resources, several thousands of objects from research and documentation, with several ten thousand attributes and references. It is possible to combine any information, for example, computing a map animation varying in time, showing the development of volcanological or impact features.

D. Case lessons learned

- Impact features have been reduced by morphological processes and are mostly only available above sea-level.
- Volcanic features are well known above and below sea-level and are more often long-term processes.
- Known impact features show a concentration in highly populated and industrialised areas.
- Both impact and volcanological features are related to social and archaeological findings.
- Both impact and volcanological features are publicly known.
- Compared with impacts and volcanological features, archaeological sites and results are not known to the same amount in order to protect the sites.

E. Classification groups

Inserting an additional object classification can extend the range of objects and disciplines. Two small examples from the context of volcanology and meteorites will show the effect on terrestrial to extra-terrestrial result matrices.

Including non-terrestrial volcanism will further lead to special object carousels. The references on “classification” of

volcanological features and distribution also explicitely leads to extraterrestrial volcanism [45]. Therefore, volcanological features on Venus from Magellan data provided from this source holds additional objects: Shield fields, intermediate volcanoes, large volcanoes, calderas, coronae, arachnoids, novae, lava floods, lava channels. Besides the above features, the volcanological and magmatic features further create references to: Vents, fields, intermediate volcanoes, steep-sided domes, tadpoles, sinuous channels, lava flows, amoeboids, festoons, and so on.

Comparable, including non-terrestrial meteorites will further lead to special object carousels, e.g., delivering data for craters on Moon and Mars.

F. Reference chains and historical knowledge

It is well known that publication references and citations are important. Besides publications, why are secondary and tertiary references on knowledge objects important? Secondary and tertiary references are as well a reliable and stable means of documentation and as well a dynamical means of optimising knowledge discovery workflows.

In this context, secondary references are links and direct references to other objects. Tertiary references are explicit or non explicit references within sets of objects.

These references are stable because they document a certain state within space and time. At the same time these references have dynamical features because their content and context can be used dynamically with workflows as well as they can refer to dynamically handled content and context.

A practical example workflow using the information on Gottfried Wilhelm Leibniz (1646–1716) in different context shows how to integrate certain historical knowledge with the knowledge resources on natural sciences in order to extend the range of discovery. The following is not a simple 'search example' but a successful integration of valuable but weakly structured knowledge data and information into advanced knowledge resources. The results are advanced discovery and research facilities.

Data referring to the "Leibniz archives" [46] has been used for extending the reference chain. The archives are a valuable source of information, which can be used to extend workflows based on knowledge resources in the future. The result matrix for "Saturn" has been computed via the workflow chain including the "Leibniz" objects including link and keyword context for creating intermediate result matrices. This includes references from concept glossaries [15] with transcriptions and scans as well as the LeibnizCentral manuscript collections and catalogues [17].

In this example, the "Saturn Rings", Leibniz, and early discoverers and scientists have up to now not been interlinked by any other available knowledge resource or referenced by any published documentation. The following methodology has been used for creating a result matrix. Links have been created by selecting the maximum time range of Leibniz dates and possible research topics and searching for research topics of other researchers in that time interval.

An algorithm supports UDC classification, building groups of pseudonyms, translations, corrections, and phonetic support based on computed LX Soundex codes for name selection used in context with Leibniz.

The result matrix contains references and associations to topics and researchers at the time of Gottfried Wilhelm Leibniz who were involved or engaged in research on comparable or even different topics. This allows to suggest which topics have been present and Leibniz may have been recognised or even which he might not have known of. Listing 3 shows an excerpt from the keyword context data of a 'Leibniz'-object "Saturn".

```
...
link-Context: LNK :: http://www.uni-muenster.de/Leibniz/DatenII2/II2_B.pdf
keyword-Context: KYW :: planetae (Planeten), Venus
...
link-Context: LNK :: http://www.uni-muenster.de/Leibniz/DatenII2/II2_B.pdf
keyword-Context: KYW :: planetae (Planeten), Saturn
...
link-Context: LNK :: http://www.uni-muenster.de/Leibniz/DatenII3/II3_B.pdf
keyword-Context: KYW :: planetae (Planeten), Saturn
...
link-Context: LNK :: http://www.bbaw.de/bbaw/Forschung/Forschungsprojekte/leibniz_potsdam/bilder/IV6text.pdf
link-Context: LNK :: http://www.nlb-hannover.de/Leibniz/Leibnizarchiv/Veroeffentlichungen/III7A.pdf
keyword-Context: KYW :: Saturn, Ring
...
```

Listing 3. Keyword context data from 'Leibniz'-object "Saturn" (excerpt).

The references resolve to the secondary data at the TELOTA service [47], using the terms "Saturn" [48] and "Planet" [49]. The links to the references are provided by different institution or respective domains [50], [51], [52]. The `II3_B` link fails as this information is not available, which is true for more than 20 other links, too, referring to this resource from this request. Even the other references from this service require additional support as they differ essentially regarding their bibliographic data, which is missing in some cases, as well as their scheme is not consistent.

Anyhow, with the correction, classification, and context support of the knowledge resources an interesting example, which has been resolved from the LX Foundation Scientific Resources is the discovery of the separation of the rings of the planet Saturn, for which sources exist documenting that the separation has been detected and recognised by Guiseppe Campani in the year 1664, about ten years before it has been published by Giovanni Domenico Cassini.

Listing 4 shows an excerpt of a secondary citation reference set used with UDC classified knowledge objects.

```
Saturn                [Astronomie, ...]:
                      Sixth planet from the sun, a gas giant or Jovian planet.
                      Saturns' most prominent feature is the Saturn ring system.
                      Guiseppe Campani detected and recognised the separation of the Saturn rings in the year \isodate{1664}{}{}.
                      %%IML: cite: NO  16640000 {LXK:Saturn; Saturn ring system; solar system; planets; discovery} {UDC:...} {PAGE:----..----} LXCITE://Campani:1664:Saturn
                      %%IML: cite: NO  20070000 {LXK:Saturn; Saturn ring system; solar system; planets; discovery} {UDC:...} {PAGE:----..----} LXCITE://Oberschelp:2007:Campani
```

```
7 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%%IML:␣cite:␣NO␣␣20110000␣{
  LXK:Saturn;␣Saturn␣ring␣system;␣solar␣system;␣planets;␣
  discovery}␣{UDC:...}␣{PAGE:----..----}␣LXCITE://
  Oberschelp:2011:Campani
8 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Giovanni␣Domenico␣Cassini␣later␣
  described␣the␣separation␣of␣the␣Saturn␣rings.
9 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%%IML:␣cite:␣NO␣␣16740000␣{
  LXK:Saturn;␣Saturn␣ring␣system;␣solar␣system;␣planets;␣
  description}␣{UDC:...}␣{PAGE:----..----}␣LXCITE://
  Cassini:1674:Saturn
10 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%%IML:␣cite:␣NO␣␣20070000␣{
  LXK:Saturn;␣Saturn␣ring␣system;␣solar␣system;␣planets;␣
  description}␣{UDC:...}␣{PAGE:----..----}␣LXCITE://
  Oberschelp:2007:Campani
11 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%%IML:␣cite:␣NO␣␣20110000␣{
  LXK:Saturn;␣Saturn␣ring␣system;␣solar␣system;␣planets;␣
  description}␣{UDC:...}␣{PAGE:----..----}␣LXCITE://
  Oberschelp:2011:Campani
```

Listing 4. Secondary citation reference set excerpt used with the UDC classified knowledge object "Saturn" (LX resources).

The secondary references from the knowledge resources refer to bibliographic objects, which resolve to [53] and [54]. From these references the tertiary references, in this case non-explicit references, refer to the knowledge resources documentation on Guiseppe Campani.

As shown, using the available features, e.g., the context categorisation from the knowledge resources it is possible to catch this information and to drastically increase the spectrum of gathering information and complementing the result matrix. The workflows and algorithms presented here can be used in order to overcome missing links in between different information pools and complement knowledge resources and workflows.

For a sustainable use of external "secondary" information any localisation and references data have to be long-term or medium-term persistent.

Currently, at least medium-term available methods have to be provided for consistent and reliable resources. This includes that the secondary data providers have to support at least concepts like URN (Uniform Resource Name), PURL (Persistent Uniform Resource Locator), DOI (Digital Object Identifier) instead of pure URL (Uniform Resource Locator).

In addition, for enabling citations and references, bibliographical data must be explicitly available as such with each reference. For long-term use this data it must be automatable, machine readable, and documented.

The current structure of the Leibniz archives' resources is explicitly not suitable for automated citation, referencing, and reuse. The sources themselves are currently distributed at several hosting institutions (e.g., Berlin, Hannover, Göttingen, Münster). For cases like these, it is recommended to implement a sustainable structure consistently over all the participated sources and to provide persistent references. Failures on the generated or provides references as shown above should be avoided in any case. All these aspects are issues, which should be seriously worked on by the Leibniz archives in order to create a sustainable future solution.

G. Flexible support for HEC and dynamical discovery

The KOA architecture is based on a flexible documentation and development architecture [29] and integrated with the case study implementations based on the Collaboration house framework for disciplines, services, and resources [28]. For the various HEC scenarios a flexible, scalable, and dynamical network solution, e.g., Software Defined Networks (SDN) is highly recommendable [55]. Building the tree paths as well as the discovery of connections in the carousels can be done in parallel, comparable to a modelling process. This way, while computing one tree it is possible to follow connections into other disciplines' branches interesting for a workflow. The task parallel processes can be computed to look ahead, dynamically discovering fitting relations. On the other hand, it is possible to compute multiple trees and create intermediate result matrices, which can be used for building multi-disciplinary results from a large number of trees.

H. Dynamical referencing

Referring objects for publicly available information can be integrated by dynamically building associations from the knowledge resources as has currently been done with search engine content, e.g., results from Google or other dynamical sources. Table VI shows the results of a Google search [56] done for the keywords "volcano, udc, classification". The results contain the UDC classification found with the request as well as the terms associated with this in the text.

Table VI. VOLCANO RESULTS FROM PUBLICLY AVAILABLE INFORMATION, GOOGLE SEARCH, STATUS OF JANUARY 2013 (EXCERPT).

UDC Classification	*In-text Terms*
551.442(437.6)	Volcano, phreatic
631.4	Volcano
553.405	Uranium, deposit, volcano
551.31:551.44(532)	Volcano
(*764)	Volcano
(*7)	Volcano

Table VII shows the results of a Google search done for the keywords "cenote, udc, classification".

Table VII. CENOTE RESULTS FROM PUBLICLY AVAILABLE INFORMATION, GOOGLE SEARCH, STATUS OF JANUARY 2013 (EXCERPT).

UDC Classification	*In-text Terms*
930.85(726.6) 551.435.8:528.9	Sinkhole cenote maya
551.44	Doline, sinkhole
556.34:519.216	Sinkhole, drainage
551.435.82(234.41)	Sinkhole, collapse
624.153.6:699.8:551.448	Sinkhole, collapse
551.44(450.75)	Karst, sinkhole, collapse
551.44(045)=20	Groundwater, surfacewater
551.44:001.4	Grotte, Höhle
551.44(450.75)	Karst, Apulia
551.44(437.2)	Geology, karst

All documents found from public external sources in this context have been identified to contain academical and scientific content. Even as this example is intended to provide a lower depth of knowledge mapping than available in specialised knowledge resources, it provides an excellent spectrum of related information for the respective disciplines. There

should be emphasis on the fact that this kind of classification on material in manually added to the content by the creator. After considering material of this kind for a knowledge discovery process an automated classification can be computed from the content independently by any service. Both types of classification can contribute in order to obtain a case-optimised result matrix at any step within the discovery workflow.

I. Workflow and computation demands

Table VIII shows the resulting computation times (wall clock) for straight and broadened application qualities. per workflow instance and request. Requests are restricted to three initial terms. Straight means calculating the result matrix directly from the plain data available, including ranking. Broadened means using full text, references, and available secondary context information, with a wide spectrum of topics. It is possible to flexibly support the knowledge discovery workflow by any number and kind of algorithms and communication. In this case classification, keywords, synonyms, phonetic algorithms, homophones, and category lists have been used.

Table VIII. STRAIGHT AND BROADENED (SERIAL) APPLICATION QUALITIES AND COMPUTATION TIMES PER WORKFLOW INSTANCE AND REQUEST (RESTRICTED TO THREE INITIAL TERMS).

Item	*Straight*	*Broadened*
Number of terms (restricted for demo.)	3	3
Comparisons	$\approx 90,000$	$\approx 1,090,000$
Selection processes	1,540	16,700
Intermediate results	420	5,100
Final results (selected top 10)	10	10
Classification evaluation time share	3 s	30 s
Keyword extraction time share	2 s	4 s
Fulltext support time share	4 s	22 s
Reference support time share	1 s	3 s
Phonetic support time share	3 s	8 s
Instance computation time	3 s	120 s

The example demonstrates the principle and tendency. Starting a single workflow instance with a small number of 3 object terms (Figure 5), this statistically results in:

a) *Straight*: Retrieval followed by 90,000 comparisons, delivers 30,000 results, ranked to create a top 10.
b) *Broadened*: This requires an additional 1 million comparisons per term and some 10,000 comparisons on more than one term as well as on subterms, it delivers 90,000 results, which are ranked to create a top 10.

In an average of terms, b) results in 3 new top terms better reflecting the context, which means a significant improvement of the quality of the result matrix. As Table VIII shows, when improving the quality, the compute time increases from about 3 seconds to 2 minutes. Over the time the resource usage increases by about a factor of 50. Due to the structure of the compute algorithms a part of the workflow processes can be done in parallel before the final result matrix is created. Other advanced workflow processes, e.g., those processes where all the intermediate results must be available before any decision on the next step can be done, have to be chained for the purpose of improving the quality. With parallel processing in the above example the overall time can be reduced to about 30 s on the same architecture if an increased number of resources is available. Increasing the number of comparisons by adding further sources for improving the quality of results increases the requirements on resources more than linear referring to the compute time. This is going ahead with a smaller amount of numerical improvement for the top results. The knowledge resources fully support this procedure. The broadened serial and task parallel (dual-core processors) application qualities per workflow instance and request are summarised in Table IX.

Table IX. BROADENED SERIAL AND PARALLEL APPLICATION QUALITIES PER WORKFLOW INSTANCE AND REQUEST (AS ABOVE).

Workflow Item	*Broadened Serial*	*Broadened Parallel*
Number of terms (restricted for demo.)	3	3
Parallel resources (nodes)	1	10
Instance computation time	120 s	20 s

The resulting computation times per instance can be efficiently reduced exploiting parallelity of resources. Modularising the knowledge resources into dynamical entity groups of objects is very efficient for a large number of requests and resources available. This is especially interesting for any wider economical and practical interactive use. The higher the complexity of the single, even non-linear, workflow is, the less efficient are todays resources architectures.

X. EVALUATION AND DISCUSSION

The results from the work and case studies presented in this paper have shown that for a higher understanding of the contributions the various implementations cannot be separated but should be considered complementary. The implemented

... concepts and structures correspond with the methodology and systematics.
... content presents the results achieved by precise sciences and their application.
... means for documentation, discovery, and computation are done exemplarily for the knowledge resources.
... universal classification views and references show the logical integration.
... components show the flexibility and extendability.
... case studies show the practical use.

The long-term multi-disciplinary focus has been sustainability, expenses and benefits, and complementary results.

A. Sustainability

Regarding the sustainability of the knowledge resources support it has been practical to consider three main aspects for creating sustainable KOA architectures.

1) *Scalability and efficiency:* The workflow process can be modularised and therefore can be implemented as scalable and parallelised algorithms.

2) *Discovery and content:* Big amounts of multi-disciplinary information will always have to consider inhomogeneous groups of information. With the described method the barrier between the inhomogeneous content, for example, between different disciplines can be overcome. The knowledge resources support structuring and modularising the workflows to a defined level. Any references that might not already exist explicitely in the knowledge resources can be suggested by a non-tree link. An example is, computing full text comparisons between the carousels from the available plain content of the knowledge resources.
3) *Universal multi-disciplinarity:* The knowledge resources allow any number of dimensional space. Besides that, the knowledge resources allow to use multi-disciplinary clustering of objects, e.g., clustering of stones for an archaeological view as well as for a petrographical view.

These features can be used for a flexible dynamically guided discovery. Besides the benefits of very flexible classification support, e.g., via UDC, expenses are that the creation and operation do require intensive work.

B. Expenses and benefits

Classification support, e.g., via UDC, does require intensive work, is expensive, and very much profits from professional experiences. The application of UDC with complex knowledge needs flexibility of the resources as wells as a flexible handling in extendability. The challenges with the distributed use of UDC are, e.g., the use of private catalogues, like external codes or author abbreviations. In addition, the sustainability of knowledge objects will benefit from the use of methods like facetted versioning, universal dates (e.g., ISO dates), and georeferencing.

C. Complementary results

With the presented object carousels an undefined number of practical workflows can be created on the knowledge resources. Examples, which have been investigated for gathering complementary results are regular expression and string search, classification search (UDC), keyword search, sort support search, references search or phonetic search (Soundex).

D. Information and classification

Text information and classification information are complementary. This is important for knowledge resources as well a for application scenarios, e.g., search algorithms. Using classification supported search algorithms can improve the result drastically. The quality of results improves from below five percent to up to over ninety percent.

XI. Conclusion and future work

Structuring and classification with long-term knowledge resources and UDC support have successfully provided efficient and economic base for an integration of multi-disciplinary knowledge and IICS components, supporting archaeological and geoscientific information systems. With these, the solution is scalable, e.g., regarding references, resolution, and view arrangements. The concept can be transferred to numerous applications in a very flexible way. The overall results on object carousels and Q4Q workflows from the implementation and case studies are:

- The quality of data can be most efficiently improved at the knowledge resources components.
- The quantity of data can be increased by referencing and intelligent discovery workflow algorithms.
- The quantity of compute and storage resources is both tightly linked with the quality of data and the quantity of data and resources requirements.

The knowledge resources can be integrated into a steadily improving system architecture storing data for successful creation of sustainable workflow definitions, meaning that the result matrix of requests can be stored for future use and evaluation. This can be done in a non-incremental way, depending on the environment of communication, computation, and storage resources in order to provide an efficient solution. Separate snapshots of the knowledge resources allow to consider developments within time. Nevertheless, for service operation this can result in very high computational requirements for resources.

Integration of external information resources has shown huge benefits on the quantity of data. The quantity can be used for creating higher quality result matrices and improve the discovery. A number of recommendations have been given for integrating external data into advanced knowledge resources. Future developments of the external resources should consider to comply with a systematic, consistent, and well structured base for their data, interfaces, and publications.

With the presented object carousels an undefined number of practical workflows can be created on the knowledge resources. The object carousels concept is part of the “tooth system” for long-term documentation and algorithms and the exploitation of supercomputing resources for use with future IICS. Work has been done [57], [58] in order to facilitate that future architectures and components will support intelligent system components and processing modelling of complex environments.

Acknowledgements

We are grateful to all national and international partners in the GEXI cooperations for the innovative constructive work. We thank the Science and High Performance Supercomputing Centre (SHPSC) for long-term support of collaborative research since 1997, including the GEXI developments and case studies on archaeological and geoscientific information systems. Special thanks go to the scientific colleagues at the Gottfried Wilhelm Leibniz Bibliothek (GWLB) Hannover, especially Dr. Friedrich Hülsmann, for prolific discussion within the “Knowledge in Motion” project, for inspiration, and practical case studies. Many thanks go to the scientific colleagues at the Leibniz Universität Hannover, especially Mrs. Birgit Gersbeck-Schierholz, to the Institute for Legal Informatics

(IRI), Leibniz Universität Hannover, and to the Westfälische Wilhelms-Universität (WWU), for discussion, support, and sharing experiences on collaborative computing and knowledge resources and for participating in fruitful case studies as well as to the participants of the postgraduate European Legal Informatics Study Programme (EULISP) for prolific discussion of scientific, legal, and technical aspects over the last years.

References

[1] C.-P. Rückemann, "Sustainable Knowledge Resources Supporting Scientific Supercomputing for Archaeological and Geoscientific Information Systems," in Proceedings of The Third International Conference on Advanced Communications and Computation (INFOCOMP 2013), November 17–22, 2013, Lisbon, Portugal. XPS Press, 2013, pp. 55–60, ISSN: 2308-3484, ISBN: 978-1-61208-310-0, URL: http://www.thinkmind.org/download.php?articleid=infocomp_2013_3_20_10034 [accessed: 2014-01-01].

[2] "WDL, World Digital Library," 2014, URL: http://www.wdl.org [accessed: 2014-01-12].

[3] "The Digital Archaeological Record (tDAR)," 2014, URL: http://www.tdar.org [accessed: 2014-01-12].

[4] Wissenschaftsrat, "Übergreifende Empfehlungen zu Informationsinfrastrukturen, (English: Spanning Recommendations for Information Infrastructures)," Wissenschaftsrat, Deutschland, (English: Science Council, Germany), Drs. 10466-11, Berlin, 28.01.2011, 2011, URL: http://www.wissenschaftsrat.de/download/archiv/10466-11.pdf [accessed: 2013-08-09].

[5] di Maio, P., "A Global Vision: Integrating Community Networks Knowledge," Community Wireless Symposium, European Community, EC Infoday, Barcelona, 5th October 2012, 2012, URL: http://people.ac.upc.edu/leandro/misc/CAPS_Paola.pdf [accessed: 2014-01-12].

[6] "Geo Exploration and Information (GEXI)," 1996, 1999, 2010, 2013, 2014, URL: http://www.user.uni-hannover.de/cpr/x/rprojs/en/index.html#GEXI [accessed: 2014-01-12].

[7] C.-P. Rückemann, Queueing Aspects of Integrated Information and Computing Systems in Geosciences and Natural Sciences. InTech, 2011, pp. 1–26, Chapter 1, in: Advances in Data, Methods, Models and Their Applications in Geoscience, 336 pages, ISBN-13: 978-953-307-737-6, DOI: 10.5772/29337, OCLC: 793915638, DOI: http://dx.doi.org/10.5772/29337 [accessed: 2014-01-12].

[8] C.-P. Rückemann, "Implementation of Integrated Systems and Resources for Information and Computing," in Proceedings of The International Conference on Advanced Communications and Computation (INFOCOMP 2011), October 23–29, 2011, Barcelona, Spain, 2011, pp. 1–7, ISBN: 978-1-61208-009-3, URL: http://www.thinkmind.org/download.php?articleid=infocomp_2011_1_10_10002 [accessed: 2014-01-12].

[9] I. Nonaka and H. Takeuchi, The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation. Oxford University Press, Oxford, 1995, ISBN: 0195092694.

[10] C. W. Holsapple and K. D. Joshi, "Knowledge Management Support of Decision Making, Organizational knowledge resources," Decision Support Systems, vol. 31, no. 1, May 2001, pp. 39–54, ISSN: 0167-9236, Elsevier, URL: http://dx.doi.org/10.1016/S0167-9236(00)00118-4 [accessed: 2013-08-10].

[11] "Universal Decimal Classification Summary (UDCS) Linked Data," 2014, URL: http://udcdata.info [accessed: 2014-01-12].

[12] "LX-Project," 2014, URL: http://www.user.uni-hannover.de/cpr/x/rprojs/en/#LX (Information) [accessed: 2014-01-12].

[13] E. W. von Tschirnhaus, "Brief (Letter), Ehrenfried Walther von Tschirnhaus an Leibniz 17.IV.1677," pp. 59–73, 1987, Gottfried Wilhelm Leibniz, Sämtliche Schriften und Briefe, Mathematischer, naturwissenschaftlicher und technischer Briefwechsel dritte Reihe, zweiter Band, 1667 – 1679, Leibniz-Archiv der Niedersächsischen Landesbibliothek Hannover, Akademie-Verlag Berlin, 1987, herausgegeben unter Aufsicht der Akademie der Wissenschaften in Göttingen; Akademie der Wissenschaften der DDR.

[14] G. F. von Franckenau, "Brief (Letter), Georg Franck von Franckenau an Leibniz 18. (28.) September 1697, Schloss Frederiksborg, 18. (28.) September 1697," pp. 568–569, Gottfried Wilhelm Leibniz Bibliothek (GWLB), Leibniz-Archiv der Niedersächsischen Landesbibliothek Hannover, URL: http://www.gwlb.de/Leibniz/Leibnizarchiv/Veroeffentlichungen/III7B.pdf [accessed: 2013-05-26] .

[15] Berlin-Brandenburgische Akademie der Wissenschaften, "Leibniz Reihe VIII," 2014, Glossary, Concepts, BBAW, Berlin, URL: http://leibnizviii.bbaw.de/glossary/concepts/ [accessed: 2014-01-12] (concepts glossary), URL: http://leibnizviii.bbaw.de/Leibniz_Reihe_8/Aus+Otto+von+Guericke,+Experimenta+nova/LH035,14,02_091v/index.html [accessed: 2014-01-12] (transcription), URL: http://leibnizviii.bbaw.de/pdf/Aus+Otto+von+Guericke,+Experimenta+nova/LH035.14,02_091v/LH035,14!02_091+va.png [accessed: 2014-01-12] (scan).

[16] Gottfried Wilhelm Leibniz Bibliothek (GWLB), Niedersächsische Landesbibliothek, "GWLB Handschriften," 2014, hannover, URL: http://www.leibnizcentral.de [accessed: 2014-01-12].

[17] "LeibnizCentral," 2014, URL: http://www.leibnizcentral.com/ [accessed: 2014-01-12].

[18] M. Fogel, "Brieffragmente (Letter fragments) about 16xx, Historici Pragmatici universal, Terrae motus, Physica," manuscript ID: 00016293, Source: Gottfried Wilhelm Leibniz Bibliothek (GWLB), Niedersächsische Landesbibliothek, GWLB Handschriften, Hannover, URL: http://www.leibnizcentral.de [accessed: 2014-01-12].

[19] M. Fogel, "Brieffragmente (Letter fragments) about 16xx, Terrae Motus in Nova Francia," manuscript ID: 00016278, Source: Gottfried Wilhelm Leibniz Bibliothek (GWLB), Niedersächsische Landesbibliothek, GWLB Handschriften, Hannover, URL: http://www.leibnizcentral.de [accessed: 2014-01-12].

[20] Gottfried Wilhelm Leibniz Bibliothek Hannover, "Collection of Copperplates," 2014, URL: http://echo.mpiwg-berlin.mpg.de/content/copperplates [accessed: 2014-01-12].

[21] N. Seeländer, "Dens animalis marini Tidae prope Stederburgum e colle limoso effossi, Figura Sceleti prope Qvedlinburgum effossi," about 1716, Copperplate, (Kupferstichplatten), printed in "Leibniz, Protogaea, Tab. XII", re-printed in "Leibniz, Opera omnia, studio L. Dutens, 1768 – Wallmann, Abhandlung von den schätzbaren Alterthümern zu Quedlinburg, 1776, Tafel S. 39", URL: http://echo.mpiwg-berlin.mpg.de/ECHOdocuView?url=/mpiwg/online/permanent/echo/copperplates/Leibniz_cup4/pageimg&start=41&pn=73&mode=imagepath [accessed: 2013-05-26].

[22] C.-P. Rückemann and B. F. S. Gersbeck-Schierholz, "Object Security and Verification for Integrated Information and Computing Systems," in Proceedings of The Fifth International Conference on Digital Society (ICDS 2011), Proceedings of The International Conference on Technical and Legal Aspects of the e-Society (CYBERLAWS 2011), February 23–28, 2011, Gosier, Guadeloupe, France / DigitalWorld 2011. XPS, 2011, pp. 1–6, ISBN: 978-1-61208-003-1, URL: http://www.thinkmind.org/download.php?articleid=cyberlaws_2011_1_10_70008 [accessed: 2013-05-26].

[23] Max Planck Institute for the History of Science, Max-Planck Institut für Wissenschaftsgeschichte, "European Cultural Heritage Online (ECHO)," 2014, Berlin, URL: http://echo.mpiwg-berlin.mpg.de/ [accessed: 2014-01-12].

[24] "CLImatological database for the World's OCeans (CLIWOC)," 2014, Climatological Database for the World's Oceans 1750–1850, URL: http://pendientedemigracion.ucm.es/info/cliwoc/ [accessed: 2014-01-01].

[25] C.-P. Rückemann, "High End Computing Using Advanced Archaeology and Geoscience Objects," International Journal On Advances in Intelligent Systems, vol. 6, no. 3&4, 2013, pages 235–255, ISSN: 1942-2679, LCCN: 2008212456 (Library of Congress), OCLC: 826628364, URL: http://www.thinkmind.org/download.php?articleid=

intsys_v6_n34_2013_7 [accessed: 2014-01-12], URL: http://lccn.loc.gov/2008212456 (LCCN Permalink) [accessed: 2014-01-12].

[26] C.-P. Rückemann, "Integrating Information Systems and Scientific Computing," International Journal on Advances in Systems and Measurements, vol. 5, no. 3&4, 2012, pp. 113–127, ISSN: 1942-261x, LCCN: 2008212470 (Library of Congress), URL: http://www.thinkmind.org/index.php?view=article&articleid=sysmea_v5_n34_2012_3/ [accessed: 2014-01-01], URL: http://www.iariajournals.org/systems_and_measurements/sysmea_v5_n34_2012_paged.pdf [accessed: 2014-01-01].

[27] "ROA, (resource-oriented architecture), (Wikipedia)," 2014, URL: http://en.wikipedia.org/wiki/Resource-oriented_architecture [accessed: 2014-01-12].

[28] C.-P. Rückemann, "Enabling Dynamical Use of Integrated Systems and Scientific Supercomputing Resources for Archaeological Information Systems," in Proceedings of The International Conference on Advanced Communications and Computation (INFOCOMP 2012), October 21–26, 2012, Venice, Italy. XPS, 2012, pp. 36–41, ISBN: 978-1-61208-226-4, URL: http://www.thinkmind.org/download.php?articleid=infocomp_2012_3_10_10012 [accessed: 2014-01-12].

[29] C.-P. Rückemann, "Advanced Scientific Computing and Multi-Disciplinary Documentation for Geosciences and Archaeology Information," in Proc. of The Int. Conf. on Advanced Geographic Information Systems, Applications, and Services (GEOProcessing 2013), February 24 – March 1, 2013, Nice, France. XPS Press, 2013, pp. 81–88, ISSN: 2308-393X, ISBN: 978-1-61208-251-6, URL: http://www.thinkmind.org/download.php?articleid=geoprocessing_2013_4_10_30035 [accessed: 2014-01-12].

[30] "Multilingual Universal Decimal Classification Summary," 2012, UDC Consortium, 2012, Web resource, v. 1.1. The Hague: UDC Consortium (UDCC Publication No. 088), URL: http://www.udcc.org/udcsummary/php/index.php [accessed: 2014-01-12].

[31] "Creative Commons Attribution Share Alike 3.0 license," 2012, URL: http://creativecommons.org/licenses/by-sa/3.0/ [accessed: 2014-01-12].

[32] "Universal Decimal Classification Consortium (UDCC)," 2014, URL: http://www.udcc.org [accessed: 2014-01-12].

[33] "Universal Decimal Classification Summary (UDCS)," 2014, URL: http://www.udcc.org/udcsummary/php/index.php [accessed: 2014-01-12].

[34] "Universal Decimal Classification Summary (UDCS) Translations," 2014, URL: http://www.udcc.org/udcsummary/translation.htm [accessed: 2014-01-12].

[35] "Universal Decimal Classification Summary (UDCS) Translations: German," 2014, URL: http://www.udcc.org/udcsummary/php/index.php?lang=6 [accessed: 2014-01-12].

[36] "Universal Decimal Classification Summary (UDCS) Exports," 2014, URL: http://www.udcc.org/udcsummary/exports.htm [accessed: 2014-01-12].

[37] "Universal Decimal Classification Consortium (UDCC)," 2013, URL: http://www.udcc.org [accessed: 2013-04-07].

[38] "Digital Object Identifier (DOI) System, The International DOI Foundation (IDF)," 2014, URL: http://www.doi.org (Information) [accessed: 2014-01-12].

[39] N. Hemsoth, "IBM Big Data VP Surveys Landscape," Datanami, 2012, March 09, 2012, URL: http://www.datanami.com/datanami/2012-03-09/ibm_big_data_vp_surveys_landscape.html [accessed: 2012-03-18].

[40] C.-P. Rückemann, "Application and High Performant Computation of Fresnel Sections," in Symposium on Advanced Computation and Information in Natural and Applied Sciences, Proceedings of The 9th International Conference on Numerical Analysis and Applied Mathematics (ICNAAM), September 19–25, 2011, Halkidiki, Greece, Proceedings of the American Institute of Physics (AIP), vol. 1389, no. 1, 2011, pp. 1268–1271, ISBN: 978-0-7354-0956-9, OCLC: 767604605, URL: http://link.aip.org/link/?APCPCS/1389/1268/1 [accessed: 2014-01-12], URL: http://dx.doi.org/10.1063/1.3637849 (DOI) [accessed: 2014-01-12].

[41] C.-P. Rückemann, "Supercomputing Resources Empowering Superstack with Interactive and Integrated Systems," in Symposium on Advanced Computation and Information in Natural and Applied Sciences, Proceedings of The 10th International Conference on Numerical Analysis and Applied Mathematics (ICNAAM), September 19–25, 2012, Kos, Greece, Proceedings of the American Institute of Physics (AIP), vol. 1479, no. 1, 2012, pp. 873–876, ISBN: 978-0-7354-1091-6, OCLC: 767604605, URL: http://link.aip.org/link/apcpcs/v1479/i1/p873/s1 [accessed: 2014-01-12], URL: http://dx.doi.org/10.1063/1.3637849 (DOI) [accessed: 2014-01-12].

[42] C.-P. Rückemann, "High End Computing for Diffraction Amplitudes," in Symposium on Advanced Computation and Information in Natural and Applied Sciences, Proceedings of The 11th International Conference on Numerical Analysis and Applied Mathematics (ICNAAM), September 21–27, 2013, Rhodes, Greece, Proceedings of the American Institute of Physics (AIP), 2013, pp. 305–308, ISBN: 978-0-7354-1184-5, OCLC: 861348382, ISSN: 0094-243X, DOI: 10.1063/1.4825483, URL: http://link.aip.org/link/?APCPCS/1558/305/1 [accessed: 2014-01-12], URL: http://dx.doi.org/10.1063/1.4825483 (DOI) [accessed: 2014-01-12].

[43] "LX-Project," 2013, URL: http://www.user.uni-hannover.de/cpr/x/rprojs/en/#LX (Information) [accessed: 2013-05-26].

[44] B. F. S. Gersbeck-Schierholz, "Testimonies of Nature and History in the Napoli and Vesuvius Region, Italy," Media Presentation, January 2014, Hannover, Germany, 2014, URL: http://www.user.uni-hannover.de/zzzzgers/bgs_volcano.html [accessed: 2014-02-23].

[45] J. W. Head, L. S. Crumpler, J. C. Aubele, J. E. Guest, and R. S. Saunders, "Venus Volcanism: Classification of Volcanic Features and Structures, Associations, and Global Distribution from Magellan Data," JGR, vol. 97, no. E8, Aug. 1992, pp. 13 153–13 197.

[46] "Gottfried Wilhelm Leibniz Bibliothek (GWLB), Leibniz-Archiv der Niedersächsischen Landesbibliothek Hannover," 2014, URL: http://www.gwlb.de/ [accessed: 2014-01-12] .

[47] "TELOTA – The Electronic Life Of The Academy," 2014, Berlin-Brandenburgische Akademie der Wissenschaften, URL: http://telota.bbaw.de/ [accessed: 2014-01-12].

[48] TELOTA, "Gottfried Wilhelm Leibniz (Sämtliche Schriften und Briefe), Sachregister Schlagwort: Saturn," 2014, TELOTA – The Electronic Life Of The Academy, Berlin-Brandenburgische Akademie der Wissenschaften, URL: http://telota.bbaw.de/leibniziv/Sachregister/sachreg_fragen.php?aktion=schlagwort&eins=saturn&band= [accessed: 2014-01-12].

[49] TELOTA, "Gottfried Wilhelm Leibniz (Sämtliche Schriften und Briefe), Sachregister Schlagwort: Planet," 2014, TELOTA – The Electronic Life Of The Academy, Berlin-Brandenburgische Akademie der Wissenschaften, URL: http://telota.bbaw.de/leibniziv/Sachregister/sachreg_fragen.php?aktion=schlagwort&eins=planet&band= [accessed: 2014-01-12].

[50] G. W. Leibniz, "Sämtliche Schriften und Briefe," 2014, sämtliche Schriften und Briefe, Zweite Reihe, Zweiter Band, Teil 2; URL: http://www.uni-muenster.de/Leibniz/DatenII2/II2_B.pdf [accessed: 2014-01-12], Universitäts- und Landesbibliothek Münster, WWU, Münster,.

[51] G. W. Leibniz, "Sämtliche Schriften und Briefe," 2014, Sämtliche Schriften und Briefe, Vierte Reihe, Sechster Band; Rechts- und Staatswesen, URL: http://www.bbaw.de/bbaw/Forschung/Forschungsprojekte/leibniz_potsdam/bilder/IV6text.pdf [accessed: 2014-01-12], Berlin-Brandenburgische Akademie der Wissenschaften (BBAW), Berlin.

[52] G. W. Leibniz, "Sämtliche Schriften und Briefe," 2014, Sämtliche Schriften und Briefe herausgegeben von der Berlin-Brandenburgischen Akademie der Wissenschaften und der Akademie der Wissenschaften in Göttingen, Dritte Reihe, Mathematischer, naturwissenschaftlicher und technischer Briefwechsel, Siebenter Band, 2011; Gottfried Wilhelm Leibniz, Mathematischer, naturwissenschaftlicher und technischer Briefwechsel, herausgegeben von der Leibniz-Forschungsstelle Hannover der Akademie der Wissenschaften zu Göttingen beim Leibniz-Archiv der Gottfried Wilhelm Leibniz Bibliothek Hannover, Sieben-

ter Band, Juli 1696 – Dezember 1698, 2011; URL: http://www.nlb-hannover.de/Leibniz/Leibnizarchiv/Veroeffentlichungen/III7A.pdf [accessed: 2014-01-12], Niedersächsische Landesbibliothek, Hannover .

[53] R. Oberschelp, Guiseppe Campani und der Ring des Planeten Saturn, (English: Guiseppe Campani and the Ring of the Planet Saturn). Niemeyer, Hameln, 2011, Lesesaal: Erlesenes aus der Gottfried Wilhelm Leibniz Bibliothek, Heft 35, ISBN: 978-3-8271-8835-9.

[54] W. Oberschelp and R. Oberschelp, Cassini, Campani und der Saturnring, (English: Cassini, Campani, and the Saturn-Ring). Verlag Harry Deutsch, 2007, vol. 33, in: Der Meister und die Fernrohre: Das Wechselspiel zwischen Astronomie und Optik in der Geschichte, Festschrift zum 85. Geburtstag von Rolf Riekher, (English: The Master and the Telescopes: The Interplay between Astronomy and Optics in History, Festschrift for the 85th birthday of Rolf Riekher), Acta Historica Astronomiae, ISBN: 9783817118045.

[55] A. Georgi, R. Budich, Y. Meeres, R. Sperber, and H. Hérenger, "An Integrated SDN Architecture for Applications Relying on Huge, Geographically Dispersed Datasets," in Proceedings of The Third International Conference on Advanced Communications and Computation (INFOCOMP 2013), November 17–22, 2013, Lisbon, Portugal. XPS Press, 2013, pp. 129–134, ISSN: 2308-3484, ISBN: 978-1-61208-310-0, URL: http://www.thinkmind.org/download.php?articleid=infocomp_2013_6_20_10087 [accessed: 2014-01-12].

[56] "Google," 2014, URL: http://www.google.de [accessed: 2014-01-12].

[57] P. Leitão, U. Inden, and C.-P. Rückemann, "Parallelising Multi-agent Systems for High Performance Computing," in Proceedings of The Third International Conference on Advanced Communications and Computation (INFOCOMP 2013), November 17–22, 2013, Lisbon, Portugal. XPS Press, 2013, pp. 1–6, ISSN: 2308-3484, ISBN: 978-1-61208-310-0, URL: http://www.thinkmind.org/download.php?articleid=infocomp_2013_1_10_10055 [accessed: 2014-01-12].

[58] U. Inden, D. T. Meridou, M.-E. C. Papadopoulou, A.-C. G. Anadiotis, and C.-P. Rückemann, "Complex Landscapes of Risk in Operations Systems Aspects of Processing and Modelling," in Proceedings of The Third International Conference on Advanced Communications and Computation (INFOCOMP 2013), November 17–22, 2013, Lisbon, Portugal. XPS Press, 2013, pp. 99–104, ISSN: 2308-3484, ISBN: 978-1-61208-310-0, URL: http://www.thinkmind.org/download.php?articleid=infocomp_2013_5_10_10114 [accessed: 2014-01-12].

A Text Retrieval Approach to Recover Links among E-Mails and Source Code Classes

Giuseppe Scanniello and Licio Mazzeo
Universitá della Basilicata, Macchia Romana,
Viale Dell'Ateneo, 85100, Potenza, ITALY,
Email: giuseppe.scanniello@unibas.it, licio.mazzeo@gmail.com

***Abstract*—During software development and evolution, the communication among stakeholders is one of the most important activities. Stakeholders communicate to discuss various topics, ranging from low-level concerns (e.g., refactoring) to high-level resolutions (e.g., design rationale). To support such a communication, e-mails are widely used in both commercial and open source software projects. Although several approaches have been proposed to recover links among software artifacts, very few are concerned with e-mails. Recovering links between e-mails and software artifacts discussed in these e-mails is a non trivial task. The main issue is related to the nature of the communication that is scarcely structured and mostly informal. Many of the proposed approaches are based on text search or text retrieval and reformulate the link recovery as a document retrieval problem. We refine and improve such solutions by leveraging the parts of which an e-mail is composed of: header, current message, and previous messages. The relevance of these parts is weighted by a probabilistic approach based on text retrieval. The results of an empirical study conducted on a public benchmark indicate that the new approach in many cases outperforms the baselines: text retrieval and lightweight text search approaches. This paper is built on [1].**

Keywords-Information retrieval, software maintenance; traceability recovery; BM25F

I. INTRODUCTION

SOFTWARE maintenance is one of the most expensive, time consuming, and challenging phase of the development process. In contrast with software development, that typically can last for 1-2 years, the maintenance phase typically lasts for many years. It has been estimated that the costs needed to perform maintenance operations range from 85% to 90% of the total cost of a software project [2]. This is due to the fact that maintenance starts after the delivery of the first version of a system and lasts until that system is dismissed [3].

A software system is continuously changed and enhanced during the maintenance phase. Maintenance operations are carried out for several reasons [4]. Corrective, perfective, and adaptive are typical examples [5]. Independently from the kind of maintenance operation, the greater part of the cost and effort to its execution is due to the comprehension of source code [6]. Pfleeger and Atlee [7] estimated that up to 60% of software maintenance is spent on the comprehension of source code. There are several reasons that make the source code comprehension even more costly and complex and range from the size of a subject software to its overall quality. Other reasons are related to the knowledge of a subject system that is implicitly expressed in software artifacts (i.e., models, documentation, source code, e-mails, and so on) [8]. This knowledge is very difficult to retrieve and it is very often enclosed in non-source artifacts [9].

Among non-source artifacts, free-form natural language artifacts (e.g., documentation, wikis, forums, e-mails) are intended to be read by stakeholders with different experience and knowledge (e.g., managers, developers, testers, and end-users). This kind of artifacts often implicitly or explicitly references to other forms of artifacts, such as source code [10]. Linking e-mails and source code could improve the comprehension of a system and could help to understand the justification behind decisions taken during design and development. Then, links between e-mails and source code could be worthwhile within the entire software lifecycle and in software maintenance, in particular [8], [11], [12]. However, linking free-form natural language artifacts to source code is a critical task [9], [13].

Although several approaches have been proposed to recover links among software artifacts (e.g., [13], [14], [15], [16]), a very few are concerned with e-mails [17], [18]. These approaches are based on text search or text retrieval and reformulate the problem of recovering links among e-mails and source code artifacts as a document retrieval problem. We refine and improve such solutions by leveraging the parts of which an e-mail is composed of, namely the header, the current message (body, from here on), and the sentences from previous messages (quote in the following). The relevance of these parts has been weighted by means of a text retrieval probabilistic model. In particular, we implemented our solution exploiting the BM25F model [19], [20]. To assess the validity of our proposal, we have conducted an empirical study on the public benchmark presented by Bacchelli *et al.* [17].

The work presented here is based on the paper presented in [1]. With respect to this paper, we provide hereafter the following new contributions:

1) The approach has been better described;
2) Related work has been discussed and differences with respect to our approach have been highlighted;
3) An extended data analysis to strengthen the achieved results has been provided;
4) An extended discussion of results and of their practical implications has been given;
5) A prototype of a supporting tool for our approach has been described.

Structure of the paper. In Section II, we discuss related work and motivations, while we illustrate our approach in

Section III. In Section IV, we present the design of the empirical evaluation, while we discuss the achieved results and possible threats to validity in Section V. Final remarks and future work conclude the paper.

II. Related Work and Motivation

Many standards include traceability as a recommended or legally required activity for software development (e.g., IEEE Std. 830-1998 [21]). Unfortunately, in projects where traceability management is not initially a systematic part of the development process, it is very difficult to establish traceability links among software artifacts [22]. Automated traceability recovery methods deal with these problems reducing the effort to construct and maintain traceability links among software artifacts (e.g., requirements and test cases) and source code [23].

In the following, we first discuss methods that use information retrieval (IR) techniques to retrieve links among any kind of software artifact and then we focus on approaches and methods to recover links between e-mails and source code.

A. IR-Based Traceability Recovery

IR-based traceability recovery approaches use IR techniques to compare a set of source artifacts with a set of target artifacts. All the possible pairs of target and source artifacts are ranked with respect to their textual/lexical similarity that is computed using an IR method. The pairs of software artifacts and their similarity form a ranked list, which is in turn analyzed by a software engineer to establish correct and false recovered traceability links. Different IR methods can be used to compute the similarity between two artifacts. The widely used methods are: vector space model (VSM) [24] and latent semantic indexing (LSI) [25]. For example, Antoniol *et al.* [16] apply VSM and a probabilistic model to trace source code onto software documentation. The first model calculates the cosine similarity of the angle between the vectors corresponding to the source and target artifacts. The probabilistic model computes a ranking score based on the probability that a document is related to a specific source code component. It is worth noting that this model is not a text retrieval probabilistic model. The authors compare these models on two small software systems. The obtained results show that the two models exhibit similar accuracy. This study is the first one in which IR methods are applied to the problem of recovering traceability links between software artifacts.

To recover traceability links between source code and documentation, Marcus and Maletic [26] use an extension of VSM, namely LSI. The validity of their approach is assessed on the same software as those in [16]. The results indicate that LSI performs at least as well as the probabilistic model and VSM and in some cases outperforms them.

To compute the similarity between software artifacts, Abadi *et al.* [27] propose the use of the Jensen-Shannon (JS) Divergence. The authors perform a comparison among IR methods (including VSM, LSI, and JS). The experimental results indicate that VSM and JS provide best results.

Capobianco *et al.* [28] propose an IR method based on numerical analysis principles. Artifacts are represented with interpolation curves. The distance between pairs of interpolation curves indicates the similarity between the artifacts represented with these curves. The authors also report an empirical evaluation, whose results suggest that their method outperforms VSM, LSI, and JS.

A work complementary to the ones highlighted above is presented by De Lucia*et al.* [29]. The authors investigate whether the use of smoothing filters improves the performances of existing traceability recovery techniques based on IR methods. The results suggest that the use of these filters significantly improves the performances of VSM and LSI.

De Lucia *et al.* [30] analyze incremental methods for traceability recovery, namely relevance feedbacks. An empirical evaluation is conducted to assess strengths and limitations of relevance feedbacks within a traceability recovery process. The achieved results suggest that even using feedbacks a part of correct links are not retrieved. However, feedbacks mostly improve retrieval performances and can be used within an incremental traceability recovery process.

An approach that combines different orthogonal IR techniques is proposed by Gethers et al. *et al.* [31]. The used techniques are those that produce dissimilar and complementary results: VSM, JS, and Relational Topic Modeling (RTM). An empirical study on six software systems is presented to assess whether the new approach outperforms stand-alone IR methods as well as any other combination of non-orthogonal methods. The results suggest that the retrieval performances are statistically better when using the new proposed method.

B. Recovery Links between E-Mails and Source Code

Although several approaches have been proposed to recover links among software artifacts (e.g., [13], [15], [16]), only a few are concerned with e-mails [17], [18], [1]. In general, these approaches can be classified as: rule-based and IR-based.

Rule-based. All these approaches are based on text search or text retrieval and reformulate this problem as a document retrieval problem. To detect latent links between emails and source code entities hand-code specific rules (i.e., sets of regular expressions) have to be specified. These rules are in turn triggered whenever they match with a portion of email text (e.g., [10]). For example, if the identifiers in the source code repository follows the CamelCase naming convention, we basically know that each identifier is either a single or a compound name (i.e., a sequence of unseparated single names). In the case of class names, all the single names start with a capital letter. Therefore, we can define a regular expression so that every time we find a string in an e-mail of the form `Foo`, `FooBar`, `FooBarXYZ`, etc., we can mark it as a link between the source code and the e-mail. This kind of approach is computationally lightweight for small/medium corpora (e.g., repositories with a small number of e-mails) and easy to implement. Conversely, they lack of flexibility since they are strictly programming-language-dependent. Even more, they do not provide any ranking score associated with the discovered link (i.e., information about a link is binary: a link is either present or not). For example, Bacchelli *et al.* [18] define and evaluate various lightweight methods for recovering

links between source code and e-mails on their benchmark. Characteristics and naming conventions of source code were exploited. The results suggest that lightweight methods can be successfully used.

IR-based. This kind of approaches (e.g., [1]) reformulate the problem as a particular instance of the more general document retrieval problem. They use IR techniques to compare a set of source artifacts (software entities) with a set of target artifacts (e-mails). Each source code entity (e.g., the class name) is used as the query to retrieve the set of most relevant e-mails. Candidate links are then devised by inspecting the ranked list of retrieved e-mails. Relevance between any pair of source and target artifacts (i.e., source code entity and email) can be determined by their textual/lexical similarity, which is computed by using a specific IR model in conjunction with a particular term-weighting score (e.g., cosine similarity using *tf-idf* vector space model) [24]. The main advantage of IR-based approaches is that they are more flexible and associate each discovered link with a ranking score. Nevertheless, the queries for retrieving relevant emails mostly consist of the strings identifying package and/or class names, which are parsed directly from source code. This leads to "semi-structured" queries, which are used to retrieve almost "unstructured" documents (e-mails), organized in a "quasi-unstructured" way. Inverted indices built on top of *free-text* documents have proven to work good when used to answer free-text queries as well (e.g., [13]). Therefore, using keywords derived from structured source code that highly likely do not appear in the email index may lead to poor retrieval results. For example, Bacchelli *et al.* [17] evaluate and compare lightweight methods based on regular expressions with LSI and VSM. This comparison is based on their benchmark. The effectiveness of the lightweight methods and LSI and VSM is evaluated on the basis of the measures: precision, recall, and F-measure. Differently from [27], the authors observe that LSI outperforms VSM on Java systems. The results achieved with these methods are close for software written in C, PHP, and ActionScript. Furthermore, the authors show that lightweight methods outperform LSI and VSM. The interpretation of the results is that: e-mails are often referred to by name, not synonyms, and source code is rare reported in. One of the concerns related to lightweight methods is that particular configurations are needed, which depend on the system understudy. This implies that on that system is required a specific knowledge to retrieve accurate links. For lightweight methods, scalability issues could be also present when the number of e-mails increases. To deal with these concerns, IR-based link recovery should to be preferred.

III. The Approach

IR-based traceability recovery approaches reformulate traceability recovery as a document retrieval problem. We refine and improve such solutions by leveraging the parts of which an e-mail is composed of: object, body, and quote. Our approach is composed by the following steps:

1) **Creating a Corpus**. A corpus is created, so that each e-mail of a subject system will have a corresponding document in the resulting corpus. Each document has three fields: header, body, and quote.
2) **Corpus Normalization**. The corpus is normalized using a different set of techniques (e.g., stop word removal and special token elimination).
3) **Corpus Indexing**. An IR engine is used to index the corpus. Different engines work in various ways. Most of them create a numerical index for each document in the corpus. In our case, the total number of terms for each field of an e-mail is determined. This information is used in our approach. In this work, we exploited VSM and BM25F.
4) **Query Formulation**. In a typical text retrieval problem, a software engineer writes a textual query and retrieves documents that are similar to that query (e.g., [32]). Differently, in IR-based traceability recovery a set of source artifacts (used as the query) are compared with set of target artifacts (even overlapping) [16]. In this work, software entities (i.e., source code) are used as the query. The textual query is normalized in the same way as the corpus and the boolean operator "AND" is used with the individual terms of that query. The number of queries is equal to the number of source code artifacts.
5) **Ranking Documents**. The index is exploited to determine similarity measures between the queries and the documents in the corpus (i.e., the e-mails). In particular, the queries are projected into the document space generated by the IR engine on the corpus. Then lexical similarities between each query and the e-mails are computed. The pairs of source and target artifacts are ranked in descending order with respect to their lexical similarity.
6) **Examining Results**. The software engineer investigates the ranked list and classifies the pairs of source and target artifacts as true or false links.

Although all these steps are part of a baseline IR-Based Traceability Recovery approaches (e.g., [13]), we sensibly change here the steps 3 and 5. In the remainder of this section, we describe how we instantiated all the steps above. Investigating alternative instances for these steps is the subject of our future work.

A. Creating a Corpus

Each e-mail results in one document in the corpus. Each document has three well defined fields: header, body, and quote. The header field contains the subject, while the body the sentences of the current message. All the sentences from previous messages are within the quote field. In particular, it includes a chain of messages (e.g., ideas, opinions, issues, or possible solutions) exchanged among stakeholders (mostly developers) linked in the sequence in which they espoused that discussion. We also consider the quote because IR approaches produce better results when a huge amount of lexical information is available [24]. Moreover, the body and the quote fields are separately considered since the lexical information within the body is on the current focus of a discussion, while the quote field includes the text that might provide useful information on the entire discussion thread.

B. Corpus Normalization

The corpus is normalized: *(i)* deleting non-textual tokens (i.e., operators, special symbols, numbers, etc.), *(ii)* splitting terms composed of two or more words (e.g., `first_name` and `firstName` are both turned into `first` and `name`), and *(iii)* eliminating all the terms within a stop word list (including the keywords of the programming languages: Java, C, ActionScript, and PHP) and with a length less than three characters. We applied these normalization rules because they have been widely applied in IR-based traceability recovery approaches (e.g., [16]).

Splitting identifiers could produce some noises in the corpus. For example, if the name of a class is `FileBuffer`, it is possible that a software engineer talks about FileBuffer in an e-mail rather than `File` and `Buffer`. However, if the identifiers are not split the things could go from bad to worse: the class name is not in the text of the e-mails (e.g., [17] and [18]), while that name is used as the query. To deal with this issue, we apply the same normalization process on both the corpus and the queries. In addition, the "AND" operator is used to formulate each query. That is, we use the "AND" operator between each pair of words in the query (e.g., `A Sample Query` is seen as `A` "AND" `Sample` "AND" `Query`).

Differently from the greater part of the traceability recovery approaches (e.g., [13], [16]), we did not apply any stemming technique [24] to reduce words to their root forms (e.g., the words `designing` and `designer` have `design` as the common radix). This is because we experimentally observed that the use of a Porter stemmer [33] led to worse results. Also, in [17] the stemming was not used for similar reasons. Investigating alternative normalization techniques and possible combinations of them is a future direction for our work.

C. Corpus Indexing

We adopt here a probabilistic IR-based model, namely BM25F [19]. This model extends BM25 [20] to handle semistructured documents from a corpus. The BM25 model was originally devised to pay attention to term frequency and document length, while not introducing a huge number of parameters to set [34]. BM25 showed very good performances [20] and then widely used specially in web document retrieval applications [35], [36]. BM25F was successively proposed to build a term weighting scheme considering the fact that documents from a corpus can be composed of fields (e.g., [35]). Each document is in the corpus and contains information on the contained fields. Then, the fields of a document differently contribute to its representation. We used BM25F because it has been successfully used on very large corpuses [36] in terms of both scalability and quality of the retrieved documents [37]. The use of other probabilistic models (e.g., the Expectation-Maximization algorithm [38]) could lead to different results. This point is subject of future work.

The difference between "vector space" and "probabilistic" IR methods is not that great. In both the cases, an information retrieval scheme is built for considering each document as a point in a multi-dimensional geometrical space. Therefore, BM25F is based on the bag-of-words model, where each document in the corpus is considered as a collection of words disregarding all information about their order, morphology, or syntactic structure. A word could appear in different fields of the same document. In this case, that word is differently considered according to the field in which it appears. Applying BM25F, each e-mail in the corpus is represented by an array of real numbers, where each element is associated to an item in a dictionary of terms. BM25F does not use a predefined vocabulary or grammar, so it can be easily applied to any kind of corpora.

BM25F works on the occurrence of each term in the fields of all the documents in the corpus. These occurrences are used to build a *term-by-document* matrix. In the current instantiation of this step we modified the original definition of BM25F to better handle the problem at the hand. In the model a generic entry of the table is computed as follows:

$$idf(t,d) = \log(\frac{N - df(t) + 0.5}{df(t) + 0.5} + 1) * weight(t,d) \quad (1)$$

where N is the total number of documents in the corpus, while df is the number of documents where the term t appears. The weight of the term t with respect to the document d is computed by $weight(t,d)$ as follows:

$$weight(t,d) = \sum_{c\ in\ d} \frac{occurs^d_{t,c} * boost_c}{((1-b_c) + b_c * \frac{l_c}{avl_c})} \quad (2)$$

l_c is the length of the field c in the document d; avl_c is the average length of the field c in the all documents; and b_c is a constant related to the field length; and $boost_c$ is the boost factor applied to the field c. $occurs^d_{t,c}$ is the number of terms t that occur in the field c of the document d. This equation is dependent on the field and document relevance and it is similar to a mapping probability. This is because BM25F is considered a probabilistic IR-based model. Regarding the constants of the equation (1), we chose 0.75 as the value for b_c, while 1 is the boost value applied to each field (i.e., header, body, and quote). These values were experimentally chosen and are customary in the IR field [37]. In the future, we plan to automate the choice of the boost values using supervised machine learning techniques [39]. For example, we could divide the benchmark in test and training sets and then the training set could be used to determine the best boost values.

In the original definition of BM25F [36], if a term occurs in over half the documents in the corpus, the model gives a negative weight to the term. This undesirable phenomena is well established in the literature [24]. It is rare in some applicative contexts, while it is common in others as for an example in the recovery of links between e-mails and source code. In such a context, in fact, e-mails quote sentences from previous messages and then the difference among e-mails (in the same discussion thread) is not that great with respect to the terms contained. To deal with this concern, we modified the computation of idf. The adopted solution is that shown in the equation (1), which is based on that suggested in [40]. The main difference with respect to the canonical computation of idf is that 1 is added to the argument of the logarithm.

D. Query Formulation

In the traceability recovery field, source artifacts are used as the query [13]. The number of queries is then equal to the number of source artifacts. In this work, we used source code entities as the source artifacts and applied the following two instantiations for Query Formulation: *(i)* class names and *(ii)* class and package names. In both the cases, the queries are normalized in the same way as the corpus. When the textual query is composed of more than one term (e.g., `ArgoStatusBar`), the boolean operator "AND" is used with the individual terms of that query (`Argo`, `Status`, and `Bar`). This implies that all the individual terms have also to exist anywhere in the text of a document.

E. Ranking Documents

For a probabilistic IR method, the similarity score between a query with the documents in the corpus is not computed by the cosine similarity and $tf - idf$ in a vector space [41], but by a different formula motivated by probability theory [24]. In this work, we used a formula based on a non-linear saturation to reduce the effect of term frequency. This means that the term weights do not grow linearly with term frequency but rather are saturated after a few occurrences:

$$score(q,d) = \sum_{t\ in\ q} idf(t) * \frac{weight(t,d)}{k_1 + weight(t,d)} \qquad (3)$$

where q is the textual query and d is a document in the corpus. The values for $idf(t)$ and $weight(t,d)$ are computed as shown in the equations (1) and (2), respectively. The parameter k_1 usually assumes values in the interval $[1.2, 2]$. We used 2 as the value because experiments suggested that it is a reasonable value [24] to maximize retrieval performances.

F. Examining Results

A set of source artifacts is compared with set of target artifacts (even overlapping). Then, all the possible pairs (candidate links) are reported in a ranked list (sorted in descending order). The software engineer investigates the ranked list of candidate links to classify them as actual or false links.

IV. Empirical Evaluation

The presentation of the study is based on the guideline suggested in [42].

A. Definition

Using the Goal Question Metrics (GQM) template [43], the goal of our empirical investigation can be defined as follows: "Analyze the adoption of our approach for the purpose of evaluating it with respect to the links between e-mails and source code from the point of view of the researcher in the context of open source systems and from the point of view of the project manager, who wants to evaluate the possibility of adopting that approach in his/her own company."

We have then formulated and investigated the following research question: *Does our proposal outperform baseline approaches based on text search or text retrieval methods?*

We considered the following baselines in our empirical investigation:

1) **BM25F with the "OR" operator**: We apply the BM25F model and the "OR" operator in the step Query Formulation. The Corpus Indexing step is executed by considering the e-mails as composed of header, body, and quote. The only difference with respect to our proposal is that the "OR" operator is used against the "AND" operator;

2) **BM25F considering body and quote together**: We apply the BM25F model and the operators "AND" and "OR". Furthermore, the Corpus Indexing step is performed considered two fields: *(i)* header and *(ii)* body and quote together;

3) **Lucene**[1] **with "AND" and "OR" operators**: In the Corpus Indexing step, we use Lucene. It uses a combination of VSM and the Boolean model to determine how relevant a document is to a query. We here apply both the operators "AND" and "OR". Since Lucene is based on VSM, more times a query term appears in a document relative to the number of times the term appears in all the documents in the corpus, the more relevant that document to the query is;

4) **VSM**: It represents the documents in the corpus as term vectors, whose size is the number of terms present in the vocabulary. Term vectors are aggregated and transposed to form a *term-document matrix*. To take into account the relevance of terms in each document and in all the corpus, many weighting schema are available. In our empirical evaluation, we employed the *tf-idf* (term frequency - inverse document frequency) weighting;

5) **LSI**: Even for a corpus of modest size, the term-document matrix is likely to have several tens of thousand of rows and columns, and a rank in the tens of thousands as well. LSI is an extension of VSM developed to overcome the synonymy and polysemy problems [25]. SVD (Singular Value Decomposition) is used to construct a low-rank approximation matrix to the term-document matrix [44]. In LSI there is no way to enforce Boolean conditions [24];

6) **Lightweight linking technique (LLT) - case sensitive (CS)**: To reference software entities from e-mails, the names of the software entities are used as text search queries. There exists a link between a software entity and an e-mail, when there is a case sensitive match on the entity name;

[1] lucene.apache.org

7) **LLT - mixed approach (MA)**: In case the name of software entities are compounded words, they are split (e.g., `ClassName` becomes `Class Name`). The compounded words are then used for the case sensitive match on the entity name, otherwise it is used a regular expression based on class and package name;

8) **LLT - MA with regular expression (RE)**: This approach is based on that above. A different regular expression is used to better handle non-Java systems. Further details about Lightweight linking techniques can be found in [17].

The baselines from 1 to 5 are different instantiations of the recovery process shown in Section III, while the others are lightweight approaches based on regular expressions. In all the IR-based baseline approaches, with the exception of the first and second one, the corpus was indexed considering together header, body, and quote. For the baselines from 4 to 8, we used the results published in [17]. For these baselines, there were available only the results, when using the class names as the queries.

B. Planning

1) Context: Many IR-based traceability recovery approaches depend on users' choices: the software engineer analyzes a subset of the ranked list to determine whether each traceability link has been correctly retrieved. It is the software engineer who makes the decision to conclude this process. The lower the number of false traceability links retrieved, the better the approach is. The best case scenario is that all the retrieved links are correct. IR-based traceability recovery methods are far from this desirable behavior [13]. In fact, IR-based traceability recovery approaches might retrieve links between source and target artifacts that do not coincide with correct ones: some are correct and others not. To remove erroneously recovered links from the candidate ones, a subset of top links in the ranked list (i.e., retrieved links) should be presented to the software engineer. This is possible by selecting a threshold to cut the ranked list (e.g., [16], [26]).

There are methods that do not take into account the similarity between source and target artifacts: *Constant Cut Point*, it imposes a threshold on the number of recovered links, and *Variable Cut Point*, it consists in specifying the percentage of the links of the ranked list to be considered correctly retrieved. Alternative possible strategies for threshold selection are based on the similarity between source and target artifacts: *Constant Threshold*, a constant threshold is chosen, *Scale Threshold*, a threshold is computed as the percentage of the best similarity value between two vectors, and *Variable Threshold*, all the links among those candidate are retrieved links whether their similarity values are in a fixed interval. In our experiment, we used the Constant Threshold method. This is the standard method used in the literature [13]. We applied this method employing thresholds assuming values between 0 and 1. The increment used was 0.01.

For each software entity, the Query Formulation step was instantiated using either the original class name or the concatenation of class and package names. The order with which class and package names were concatenated is indifferent. We opted for these query formulations because used in [17]. Other instantiations for the Query Formulation step are possible. This point is subject of future work.

2) Variable selection: The traceability links retrieved by applying both our approach and the baselines are analyzed in terms of *correctness* and *completeness*. Correctness reflects the fact that an approach is able to retrieve links that are correct. To estimate the correctness, we used (as customary) the *precision* measure. On the other hand, completeness reflects how much the set of retrieved links is complete with respect to the all actual links. The *recall* measure is used to estimate this aspect. We used here the following definitions:

$$precision = \frac{|TP|}{|TP| + |FP|} \quad recall = \frac{|TP|}{|TP| + |FN|} \tag{4}$$

where TP (true positives) is the set of links correctly retrieved. The set FN (false negatives) contains the correct links not retrieved, while FP (false positives) the links incorrectly presented as correct ones.

When the e-mails in the benchmark do not have any reference to source code artifacts, the union of TP and FN is empty (i.e., $|TP| + |FN| = 0$). In all these cases, we cannot calculate the values for the recall measure. The values for precision could not be computed in case the approach found no link between an e-mail and the source code. Similar to [17], we avoided these issues calculating the average of $|TP|$, $|FP|$, and $|FN|$, on the entire dataset. We then computed the average values for precision and recall. Precision and recall assume values in the interval $[0, 1]$. The higher the precision value, the more correct the approach is. Similarly, the higher the recall value, the better the approach is.

To get a trade-off between correctness and completeness, we applied the balanced F-measure (F_1):

$$F_1 = \frac{2 * precision * recall}{precision + recall} \tag{5}$$

We applied this formula because we would like to emphasize neither recall nor precision. F_1 was used to estimate the *accuracy* of the approach. This is the main criterion we considered in the study. This measure has values in the interval $[0, 1]$. When comparing two approaches, the one with higher F_1 value is considered the best, namely the most accurate.

3) Instrumentation: Regarding the baselines from 1 to 3, we implemented the underlying instances of the process in a prototype of a supporting software tool. This tool was intended as an Eclipse plug-in. It can be downloaded at www.scienzemfn.unisa.it/scanniello/LASCO/. We named that plug-in LASCO (Linking e-mAils and Source Code). In the following, we highlight this tool prototype. The interested reader can find more details on LASCO in [45]. It is worth mentioning that the baselines from 1 to 3 are different instantiations of our process shown in Section III. These instantiations have been implemented in our prototype, but they are not available in the current distribution of LASCO.

In Table I, we report the steps needed to recover links between e-mail and source code (steps 4.a and 4.b) with LASCO. The steps to search e-mails that are similar to a given

TABLE I. USAGE STEPS OF LASCO

Usage Step	Expected Output	Description
1. Selecting the e-mail repository (steps 1 and 2 of the approach)	The e-mail repository is loaded.	In this step, an e-mail repository is loaded and the corpus is created. The corpus is also normalized.
2. Indexing the corpus (step 3 of the approach)	An index of the corpus is created using an IR engine.	Most of IR methods create a numerical index for each document in the corpus. LASCO supports both Lucene and BM25F (two fields, namely header, body and quote together, and three fields, namely header, body and quote). The corpus is also normalized.
3.a Recovering the links (steps 4 and 5 of the approach)	The ranked list for the system/s is computed.	The class name or package and class names are are used as the query. These names are compared with set of target artifacts: the e-mails. The textual query is normalized in the same way as the corpus. The user can use the boolean operators AND and "OR". The ranked list is then computed.
3.b Formulating a textual query (steps 4 and 5 of the approach)	The user exploits LASCO to write a textual query.	The user writes a query and the system retrieves the e-mails that are more similar to that query. The textual query is normalized in the same way as the corpus. The boolean operators AND and "OR" can be used.
4.a The ranked list is shown (step 6 of the approach)	Links between e-mails and source code are shown.	The links between e-mails and software entities (e.g., class names) are reported in a ranked list. The e-mails more similar to the software entities are shown first.
4.b The list of e-mails is shown (step 6 of the approach)	The e-mails similar to the textual query are shown.	The e-mails are reported in a ranked list. The e-mails in that list are in decreasing order with respect to their similarity with the textual query.

textual query (steps 3.b and 4.b) are reported as well. The expected output for each step is mentioned together with its description. We made also clear the connection between each step of the approach shown in Section III and the usage steps of our tool.

Figure 1 shows the top of the ranked list attained on the system Freenet. Each link in that list is characterized by a source artifact (i.e., the class name in this case) and the target artifact (i.e., the e-mail). The similarity score between these two artifact is also reported. LASCO also shows the e-mail (i.e., header, body, and quote) associated to a link double clicking that link in the ranked list. On the other hand, Figure 2 shows the screenshot that allows a user to specify a textual query (step 3.b).

To estimate the correctness, the completeness, and the accuracy of our approach and to compare it with the baselines, we used the benchmark proposed in [18]. To build the benchmark, the authors manually analyzed the e-mails of six unrelated software systems written in four different languages: Java, ActionScript, PHP, and C. 99 versions for these systems were considered. The systems are all open-source and both the code and the e-mails are freely accessible on the web. The benchmark was built on development mailing lists and considered a reliable sample set from all the e-mails in these lists. Some information on the benchmark is shown in Table II. In particular, the first column shows the name of the open source

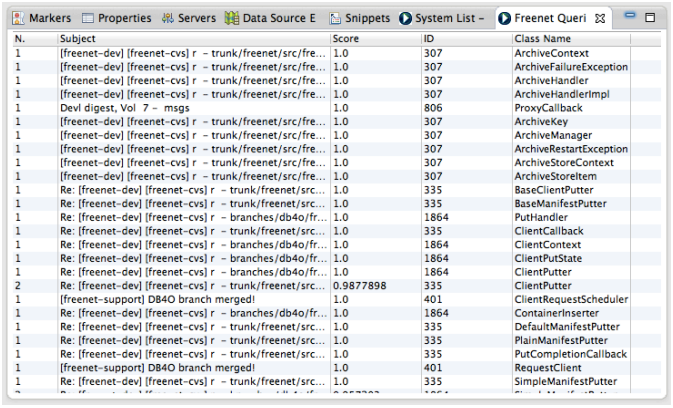


Figure 1. Ranked list attained on Freenet

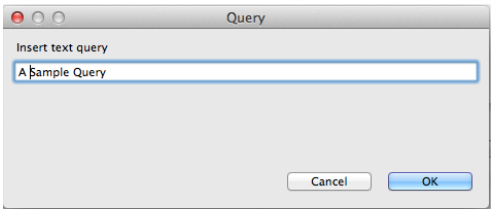


Figure 2. Formulating a textual query in LASCO

software system. A short summary of the functionality of the system and the programming language used to implement it are presented in the second and third columns, respectively. The total number of e-mails for each system are reported in the fourth column. Details on the sample are presented in the last three columns. In particular, these columns shows the size of the statistically significant sample set of e-mails, the number of e-mails in the sample with at least one reference to a software entity, and the total number of links from the e-mails in the sample. Further details can be found in [18].

For each system and all the threshold values, we computed the values of precision, recall, and F_1. To this end, we have implemented and used a tool to automatically collect TP, FP, and FN. To compare our approach with the baselines, we selected the constant threshold that produced the best accuracy. The values of precision, recall, and F_1 are computed in LASCO for our approach and the baselines from 1 to 3.

V. RESULTS AND DISCUSSION

In this section, we present and discuss the results and some lesson learned. The section concludes presenting possible threats that could affect the validity of the results.

TABLE II. SUMMARY INFORMATION ON THE USED BENCHMARK [18]

System	Language	E-mails	Sample Size	E-mails with a link	Total links
ArgoUML	Java	29,112	355	108	290
Freenet	Java	26,412	379	148	570
JMeter	Java	20,554	380	207	617
Away3D	Action Script 3	9,757	370	243	747
Habari	PHP 5	13,095	374	135	252
Augeas	C	2,219	281	140	273

TABLE III. BM25F RESULTS INDEXING THE CORPUS USING: *(i)* HEADER, *(ii)* BODY, AND *(iii)* QUOTE

	Class Name + "AND"			Class Name + "OR"			Class and Package Names + "AND"			Class and Package Names + "OR"		
System	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1
ArgoUML	0.32	0.53	0.40	0.05	0.55	0.10	0.41	0.72	0.52	0.04	0.48	0.07
Freenet	0.23	0.49	0.31	0.03	0.23	0.06	0.30	0.52	0.39	0.02	0.40	0.05
JMeter	0.32	0.41	0.36	0.10	0.41	0.16	0.49	0.62	0.55	0.06	0.43	0.10
Away3D	0.31	0.51	0.39	0.15	0.24	0.18	0.39	0.44	0.41	0.12	0.24	0.16
Habari	0.77	0.48	0.59	0.29	0.35	0.32	0.77	0.48	0.59	0.29	0.35	0.32
Augeas	0.12	0.27	0.16	0.04	0.32	0.08	0.12	0.26	0.16	0.04	0.32	0.08
Average value	0.35	0.45	0.37	0.11	0.35	0.15	0.41	0.51	0.44	0.10	0.37	0.13

TABLE IV. BM25F RESULTS INDEXING THE CORPUS USING: *(i)* HEADER AND *(ii)* BODY AND QUOTE TOGETHER

	Class Name + "AND"			Class Name + "OR"			Class and Package Names + "AND"			Class and Package Names + "OR"		
System	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1
ArgoUML	0.34	0.51	0.41	0.07	0.58	0.12	0.40	0.46	0.43	0.05	0.55	0.09
Freenet	0.22	0.54	0.31	0.08	0.45	0.14	0.29	0.62	0.40	0.07	0.5	0.13
JMeter	0.29	0.45	0.36	0.14	0.41	0.21	0.34	0.66	0.45	0.12	0.45	0.19
Away3D	0.29	0.76	0.42	0.21	0.24	0.22	0.37	0.44	0.40	0.16	0.23	0.19
Habari	0.74	0.52	0.61	0.46	0.45	0.46	0.74	0.52	0.61	0.46	0.45	0.46
Augeas	0.11	0.35	0.17	0.10	0.17	0.13	0.11	0.35	0.17	0.06	0.35	0.10
Average value	0.33	0.52	0.38	0.18	0.38	0.21	0.38	0.5	0.41	0.15	0.42	0.19

TABLE V. LUCENE RESULTS

	Class Name + "AND"			Class Name + "OR"			Class and Package Names + "AND"			Class and Package Names+ "OR"		
System	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1
ArgoUML	0.32	0.50	0.39	0.06	0.50	0.11	0.39	0.47	0.43	0.03	0.53	0.06
Freenet	0.20	0.59	0.30	0.07	0.47	0.11	0.27	0.64	0.38	0.05	0.56	0.10
Jmeter	0.27	0.46	0.34	0.10	0.36	0.15	0.34	0.70	0.46	0.07	0.49	0.13
Away3D	0.29	0.77	0.42	0.17	0.22	0.19	0.37	0.44	0.40	0.13	0.24	0.17
Habari	0.61	0.55	0.58	0.45	0.40	0.43	0.61	0.55	0.58	0.45	0.40	0.42
Augeas	0.10	0.27	0.15	0.05	0.21	0.08	0.10	0.27	0.15	0.05	0.20	0.08
Average value	0.30	0.52	0.36	0.15	0.36	0.18	0.35	0.51	0.40	0.13	0.40	0.16

TABLE VI. RESULTS BY BACCHELLI *et al.* [17]

	VSM with $tf - idf$			LSI			LLT - case sensitive			LLT - mixed approach			LLT - mixed approach with RE		
System	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1
ArgoUML	0.25	0.34	0.29	0.60	0.48	0.53	0.27	0.68	0.38	0.64	0.61	0.63	0.35	0.68	0.46
Freenet	0.15	0.25	0.19	0.62	0.43	0.51	0.17	0.70	0.27	0.59	0.59	0.59	0.27	0.69	0.39
JMeter	0.21	0.34	0.26	0.52	0.40	0.45	0.15	0.73	0.25	0.59	0.65	0.62	0.30	0.72	0.42
Away3D	0.35	0.31	0.33	0.35	0.33	0.34	0.32	0.74	0.44	0.40	0.54	0.46	0.41	0.72	0.52
Habari	0.34	0.39	0.36	0.36	0.41	0.38	0.40	0.41	0.41	0.83	0.09	0.17	0.49	0.38	0.43
Augeas	0.10	0.20	0.14	0.10	0.28	0.14	0.09	0.72	0.15	0.14	0.02	0.04	0.15	0.64	0.24
Average value	0.23	0.31	0.26	0.43	0.39	0.39	0.23	0.66	0.32	0.53	0.42	0.42	0.33	0.64	0.41

A. Results

The results achieved by applying our approach are summarized in Table III. The table also reports the results achieved by applying the "OR" operator. The results are grouped according to the two different instantiations of the step Query Formulation: *(i)* class name and *(ii)* class and package names. The last row reports the average values for each measure. Better average accuracy was achieved using class and package names and the "AND" operator (F_1 = 0.44). With respect to each individual system, we obtained the higher accuracy for Habari, namely the system implemented in PHP (F_1 = 0.59). On that system, the higher value of correctness was also obtained (precision = 0.77). It is worth mentioning that the results for that system are the same both using class name alone and class and package names together. This is because PHP 5 did not have packages. Namespaces (i.e., packages) where only introduced in PHP 5.3. The same held for Augeas (the C software system).

Table IV shows the results achieved by indexing the corpus using: *(i)* header and *(ii)* body and quote together. With respect to accuracy, better results were achieved using the operator "AND" and class and package names. The best average accuracy value was 0.41. Among the analyzed software systems, the best accuracy was obtained for Habari (F_1 = 0.61). Figure 3 summarizes the accuracy results achieved by indexing the corpus considering header, body, and quote (three fields) and header and body and quote together (two fields). This figure shows that: it is better to use the "AND" operator, the use of three fields produces better results, and formulating the query as class and package names is better.

The results achieved with Lucene are shown in Table V. The best average accuracy value was reached using the operator "AND" and class and package names (i.e., 0.40). The better accuracy was achieved for Habari (F_1 = 0.58).

These results are presented for each system in the benchmark and are grouped according to the operator used in the Query Formulation step. The average values are reported in the last row. As for BM25F, we obtained on average better results by applying the "AND" operator and using class name and

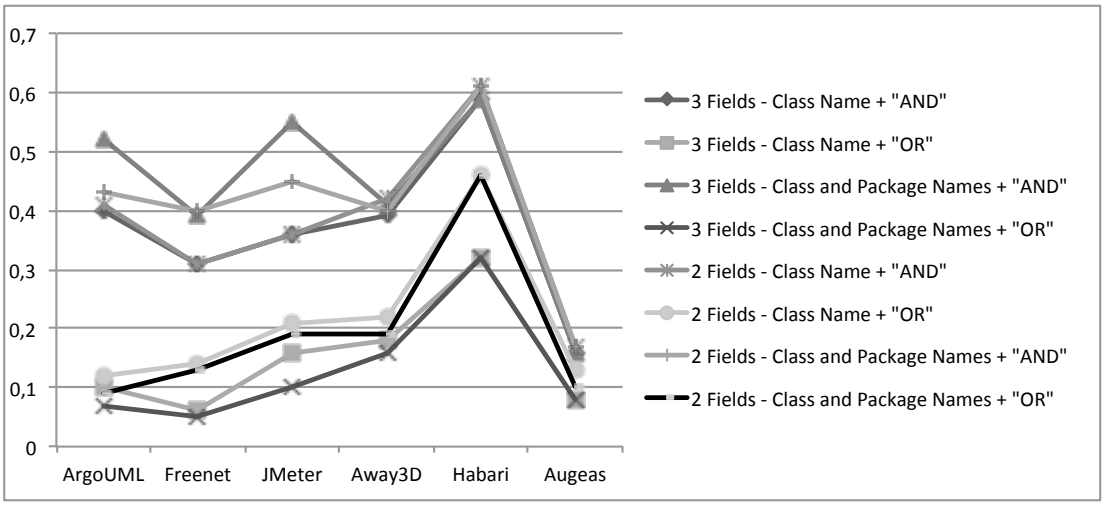


Figure 3. Accuracy results achieved with the "AND" and "OR" operators and indexing the corpus using: *(i)* header, body and quote and *(ii)* header and body and quote together

package. The comparison of these results with those achieved with our approach suggests that the use of BM25F improves the correctness and the accuracy of the retrieved traceability links. The main effect of the probabilistic model used is on the $precision$ values. The average improvement ranges from 3% to 5%. The improvement on the accuracy is on average 2% with class name and the "AND" operator, while is 1% class name and package and the "AND" operator. The use of that model does not improve completeness. The benefit deriving from the instantiation based on the probabilistic model are still better, when using the "OR" operator. The retrieved links are more precise, complete, and then accurate.

Table VI summarizes the results presented in [17], instantiating Query Formulation step with class name. As mentioned before, the results for class and package names together are not reported for VSM and LSI because the authors observed that better results were achieved using only class names. Table VI also shows the results for the lightweight linking techniques.

The results indicate that our proposal is more accurate than BM25F using two fields (header and body and quote together) on all the Java systems with the exception of Freenet (the F_1 values were 0.39 and 0.40, respectively). On the non-Java systems, the use of BM25F indexing the corpus with three or two fields did not produce remarkable differences in accuracy (see the Tables III and IV and Figure 3).

Our approach using class and package names as the queries is more accurate than VSM. Similar results were achieved for Lucene using both the operators and class name and class and package names together as the queries. Indeed, our proposal did not outperform Lucene only on Away3D when using the "AND" operator and class name as the query. The F_1 values were 0.41 and 0.42, respectively.

As far as LSI is concerned, our approach is more accurate on all the non-Java system and Jmeter. For ArgoUML the difference in favor of LSI was negligible (the F_1 values was 0.52 with respect to 0.53). A larger difference in accuracy was obtained for Freenet.

Regarding the lightweight approaches, our proposal outperformed LLT-CS in accuracy on all the systems with the exception of Away3D (the F_1 values were 0.44 and 0.41, respectively). BM25F with three fields was on average more accurate than LLT MA with and without RE (see the average values of F_1). With respect to LLT MA, we achieved better F_1 values results on Habari and Augeas (0.59 vs. 0.17 and 0.16 vs. 0.04, respectively). On the Java systems, LLT MA was more accurate than our approach. For LLT MA RE, we reached better results on the Java systems and Habari.

Regarding the correctness and completeness of the retrieved links, we can observe an interesting pattern in the data: our approach mostly allowed obtaining a more complete set of retrieved links that are correct. This result is desirable when you are interested in the recovery of links among software artifacts (e.g., [13]).

B. Discussion

Several aspects must be taken into account before drawing conclusions. We discuss IR-based approaches first and than the lightweight ones. The section concludes with our overall recommendation.

1) IR-Based recovery: For the Java systems, LSI outperformed other approaches based on IR techniques with respect to the accuracy of the retrieved links. A possible reasons is that each e-mail in the corpus quotes a large number of sentences from previous messages. This is the best scenario for using LSI [24]. In fact, this technique is used to disclose the latent semantic structure of the corpus and to recognize its topics, so dealing with synonymy and polysemy problems. Further, each document in the corpus has a large size as compared with the entities used as the queries. This might also represent another possible reason for having achieved better results for the Java systems. The considerations above and the fact that LSI outperformed our approach in terms of accuracy only on

Freenet (this difference was 0.04, while this difference was in favor of our approach on ArgoUML and JMeter and was 0.01 and 0.1, respectively) suggest that BM25F represents a viable alternative also when dealing with large documents in the corpus. It is also possible that differently tuning the parameters of our solution (e.g., the non-linear saturation) the difference with LSI could decrease on Freenet. This represents a possible future extension for our study.

In case the e-mails in the corpus quote a small number of sentences from previous e-mails, our approach outperformed other baseline approaches based on IR techniques. This happened for all the non-Java systems. For the Habari system, the e-mails were very short and then BM25F made the difference also considering the information in the body and quote together.

For the system implemented in C (i.e., Augeas), the application of the IR-Based approaches mostly produced worse results in terms of correctness, completeness, and accuracy. As also suggested in [17], a possible justification is related to the names of the entities. However, our approach outperformed the IR based baselines. Again, indexing the e-mails considering two or three fields did not produce remarkable differences.

The instantiation of the Query Formulation step with class and package names improved the correctness and completeness when our technique was used. Then, it is possible that the choice of the source artifact can make the difference in the accuracy of the links recovered. This point needs future and special conceived investigations.

The use of a stemming technique in the Normalization step produced worse results. Then, this technique seems useless in the recovery of links between source code and e-mails, when using BM25F (with two and three fields) and Lucene. On these instances, the use of the "AND" operator led to better results in terms of accuracy and correctness of the retrieved links with respect to the "OR" operator. This result held for all the systems. For completeness, the results achieved with the "AND" operator were mostly better than those achieved with the "OR" operator. Only in four cases the use of the "OR" operator led to better recall values.

The use of source code (program statements and/or source code comment) as the query was also analyzed. The results revealed that this kind of instantiation for the Query Formulation step led to worse results with respect to the other two kinds of queries considered here. This result is in line with that shown in [17] and has the following implication: it is better to use class name and class and package names as the queries.

We also performed an analysis to get indications on whether BM25F might introduce scalability issues. We used a laptop equipped by a processor Intel Core i7-2630QM with 4 GB of RAM and Windows Seven Home Premium SP-1 64bit as operating system. This analysis was performed on each system and the baseline processes implemented for our experiment (see Section IV-A). The results indicated that the time to build, normalize, and index the e-mails of the entire benchmark was twice when using three fields (i.e., 5033 milliseconds) with respect to the use of two fields (i.e., 2668 milliseconds). For Lucene, the average execution time on all the systems in the benchmark was 2660 milliseconds. For the Query Formulation step, nearly the same pattern was observed. Further details are not provided for space reason.

2) Lightweight Approaches: Regarding the accuracy of the retrieved links, LLT MA outperformed other lightweight techniques and our approach on the Java systems. On the non-Java system with the exception of Away 3D, LLT MA did not outperform our approach and the differences in the F_1 values were significant (0.59 vs. 0.17 and 0.16 vs. 0.04, respectively). The difference on Away3D was small (F_1 values were 0.41 and 0.44, respectively). Similarly, LLT MA did not outperform LLT MA RE on the non-Java systems. The achieved results suggest that our approach and LLT MA RE are more independent from the kind of documents in the corpus. Since our approach was more accurate, we can then conclude that it is the best and can be applied without making any assumption on the mailing list and the programming language of the understudy system. The same did not hold for lightweight techniques based on regular expressions because they heavily rely on common conventions and intrinsic syntactical characteristics of the corpus [17].

C. Lesson Learned

The accuracy of our approach increased when e-mails contain a huge amount of text and the entity names are carefully chosen and naming conventions are used. Furthermore, when e-mails did not contain a huge amount of text, the application of BM25F on two or three fields did not produce noteworthy differences. Then, BM25F on header, body, and quote with the operator "AND" is the best alternative (see Figure 3).

We experimentally observed that, in terms of accuracy, our approach outperformed on 5 out of 6 systems the lightweight technique that is more independent from the kind of e-mails in the corpus (i.e., LLT MA RE) [17]. To apply our approach, any assumption on the system understudy has to be made and any particular configuration setting is required. Therefore, our approach is easier to use than lightweight approaches and it is accurate enough to be worth the costs it may introduce in the corpus preprocessing and indexing phases. Furthermore, IR-based approaches, such as the one we introduce here, are more scalable. They are more efficient than lightweight techniques when the number of e-mails in the corpus increases. Finally, lightweight techniques return documents without any ranking: an e-mail either matches or not a regular expression. As a consequence, all the retrieved links have to be analyzed. To deal with this issue, text retrieval and text search techniques could be used in combination. This point could be the subject of future work.

1) Pieces of evidences: We distilled the achieved findings and lesson learned into the following pieces of evidence (PoE):

- PoE1. Accuracy increases when using class and package names as the queries;
- PoE2. Applying our approach on three fields (i.e., header, body, and quote) improves the results when the corpus contains e-mails with a huge amount of text and the entity names are carefully chosen by developers;
- PoE3. Using the "AND" operator leads to better results in terms of correctness, completeness, and accuracy;

PoE4. The corpus normalization by using stemming techniques reduces the accuracy of the recovered links;
PoE5. Our approach scales reasonable well also when the number of documents in the corpus increases;
PoE6. Our approach is more independent from the mailing list than lightweight approaches.

D. Threats to Validity

To comprehend the strengths and limitations of our study, we present here the threats that could affect the validity of the results and their generalizability. Although our efforts in mitigating as many threats as possible, some threats are unavoidable. For example, a possible threat to the validity of the results is related to the used benchmark. It is built on human judgement and then links in the benchmark could be wrong. To alleviate this threat to the construct validity, the authors of the benchmark applied several strategies (see Section 6.1 in [17]). Another threat related is that the researchers involved in the creation of the benchmark were unfamiliar with the systems and then they could have missed implicit references to software entities that an actual developer might understand. The use of a sample set of the entire dataset may also affect the validity of the results.

The use of open source software represents another threat to validity. To alleviate this threat, the benchmark was built on 6 different systems developed from separate communities and implemented in 4 programming languages based on two paradigms: object-oriented and procedural. Despite this effort, there are some differences between commercial and open source software systems. For example, open source software is usually developed outside companies mostly by volunteers and the development methodologies used are different from the ones commonly applied in the software industry. Although many large companies are using open source software in their own work or as a part of their marketed software, it will be worth replicating the study on real project. These replications will help us to confirm or contradict the achieved results.

The instantiation of the Query Formulation step represents another threat. We used class names or class and package names. The observed results suggest that this aspect influences the accuracy, correctness, and completeness of the results. In this work, we used the instantiations above to compare our approach with those in [17]. Empirical studies are needed to better assess how different kinds of queries affect the quality of the retrieved links.

Further threats concern the validity of the comparison between the results achieved with our approach and those obtained with the baselines. In the experiment presented in this paper, we could not perform statistical analyses because the results presented in [17] were not provided in an adequate form and replications were not possible (e.g., regular expressions were not described at an adequate level of detail). Although a comparison was possible between our approach and the baselines from 1 to 3, at this step of our research we preferred to propose a new approach and compare it with the lightweight approaches proposed in [17], namely the state of the art in the recovery of links between e-mails and source code.

VI. Conclusion

For software maintainers, who are unfamiliar with a software system, the recovery of links among free-form natural language software artifacts can be a laborious task if performed manually. We proposed, implemented, and evaluated an approach to recover links between e-mails and source code. The approach is based on text retrieval techniques combined with the BM25F probabilistic model. We used this model because it showed very good performances [20], [35], [36]. The defined approach is general and can be applied to software implemented with any programming language and to any kind of documents (i.e., e-mails) in the corpus. To assess the validity of our proposal, we conducted an empirical study using a public benchmark [18]. Based on this benchmark, we performed a comparison between our approach and 8 baselines. The results indicated that our approach in many cases outperformed the IR-based baseline approaches and the lightweight techniques proposed in [17].

Furthermore, our approach scales well when the number of e-mails increases and it does not require any assumption on the system understudy. Traditionally, probabilistic IR has had neat ideas but the methods have never won on performance [24]. This is possibly due to the severity of the modeling assumptions that makes achieving good performance difficult. Things changed when the BM25 weighting method was introduced. Our results confirm that in a different context.

Acknowledgment

The authors would like to thank the Michele Lanza and the Alberto Bacchelli for their support and for having made available the benchmark used in this work. Special thanks are due to the Anna Tolve and the Raffaele Branda, who developed some of the software modules of the prototype implementing the approach presented here.

References

[1] R. Branda, A. Tolve, L. Mazzeo, and G. Scanniello, "Linking e-mails and source code using bm25f," in *International Conference on Software Engineering Advances*, 2013, pp. 271–277.

[2] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT Professional*, vol. 2, pp. 17–23, 2000.

[3] M. V. Zelkowitz, A. C. Shaw, and J. D. Gannon, "Principles of software engineering and design," 1979.

[4] M. M. Lehman, "Program evolution," vol. 19, no. 1, pp. 19–36, 1984.

[5] E. B. Swanson, "The dimensions of maintenance," in *Proc. of International Conference on Software Engineering*. IEEE CS Press, 1976, pp. 492–497.

[6] A. V. Mayrhauser, "Program comprehension during software maintenance and evolution," *IEEE Computer*, vol. 28, pp. 44–55, 1995.

[7] S. Pfleeger and J. Atlee, *Software Engineering - Theory and Practice*. Pearson, 2006.

[8] A. De Lucia, F. Fasano, C. Grieco, and G. Tortora, "Recovering design rationale from email repositories," in *Proc. of the International Conference on Software Maintenance*. IEEE, 2009, pp. 543–546.

[9] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Trans. Software Eng.*, vol. 32, no. 1, pp. 4–19, 2006.

[10] A. Bacchelli, M. Lanza, and M. D'Ambros, "Miler: a toolset for exploring email data," in *Proceedings of the International Conference on Software Engineering*. ACM, 2011, pp. 1025–1027.

[11] N. Bettenburg, B. Adams, A. E. Hassan, and M. Smidt, "A lightweight approach to uncover technical artifacts in unstructured data," *International Conference on Program Comprehension*, vol. 0, pp. 185–188, 2011.

[12] N. Bettenburg, S. W. Thomas, and A. E. Hassan, "Using code search to link code fragments in discussions and source code," in *CSMR '12: Proceedings of the 16th European Conference on Software Maintenance and Reengineering*, IEEE. IEEE, 2012, pp. 319–329.

[13] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, 2007.

[14] H. Sultanov and J. H. Hayes, "Application of swarm techniques to requirements engineering: Requirements tracing," in *Proc. of IEEE International Requirements Engineering Conference*, ser. RE '10. IEEE CS Press, 2010, pp. 211–220.

[15] S. K. Sundaram, J. H. Hayes, A. Dekhtyar, and E. A. Holbrook, "Assessing traceability of software engineering artifacts," *Requir. Eng.*, vol. 15, no. 3, pp. 313–335, 2010.

[16] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Trans. Software Eng.*, vol. 28, no. 10, pp. 970–983, 2002.

[17] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proc. of International Conference on Software Engineering*. ACM, May 2010, pp. 375–384.

[18] A. Bacchelli, M. D'Ambros, M. Lanza, and R. Robbes, "Benchmarking lightweight techniques to link e-mails and source code," in *Proc. of Working Conference on Reverse Engineering*. IEEE Computer Society, 2009, pp. 205–214.

[19] S. Robertson, H. Zaragoza, and M. Taylor, "Simple bm25 extension to multiple weighted fields," in *Proc. of International Conference on Information and Knowledge Management*, ser. CIKM '04. ACM, 2004, pp. 42–49.

[20] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," *Found. Trends Inf. Retr.*, vol. 3, pp. 333–389, April 2009. [Online]. Available: http://dx.doi.org/10.1561/1500000019

[21] IEEE, *IEEE Recommended Practice for Software Requirements Specifications*, Std., 1998. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=720574

[22] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," *Computer*, vol. 40, pp. 27–35, June 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1271918.1271947

[23] G. Antoniol, G. Canfora, G. Casazza, and A. D. Lucia, "Maintaining traceability links during object-oriented software evolution." *Softw., Pract. Exper.*, vol. 31, no. 4, pp. 331–355, 2001.

[24] C. D. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[25] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society of Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[26] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proc. of the International Conference on Software Engineering*. IEEE CS Press, 2003, pp. 125–137.

[27] A. Abadi, M. Nisenson, and Y. Simionovici, "A traceability technique for specifications," in *Proc. of the International Conference on Program Comprehension*. IEEE CS Press, 2008, pp. 103–112.

[28] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Traceability recovery using numerical analysis," in *Proc. of the International Working Conference on Reverse Engineering*. IEEE CS Press, 2009, pp. 195–204.

[29] A. De Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Improving ir-based traceability recovery using smoothing filters," in *Proc. of the International Conference on Program Comprehension*. IEEE CS Press, 2011, pp. 21–30.

[30] A. De Lucia, R. Oliveto, and P. Sgueglia, "Incremental approach and user feedbacks: a silver bullet for traceability recovery," in *Proc. of the International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 299–309.

[31] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *Proceedings of the International Conference on Software Maintenance*. IEEE CS Press, 2011, pp. 133–142.

[32] V. Rajlich and N. Wilde, "The role of concepts in program comprehension," in *Proc. of the International Workshop on Program Comprehension.*, 2002, pp. 271–278.

[33] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[34] K. S. Jones, S. Walker, and S. E. Robertson, "A probabilistic model of information retrieval: development and comparative experiments," *Inf. Process. Manage.*, vol. 36, pp. 779–808, November 2000.

[35] K. Y. Itakura and C. L. Clarke, "A framework for bm25f-based xml retrieval," in *Proc. of International Conference on Research and Development in Information Retrieval*. ACM, 2010, pp. 843–844.

[36] J. R. Pérez-Agüera, J. Arroyo, J. Greenberg, J. P. Iglesias, and V. Fresno, "Using bm25f for semantic search," in *Proc. of the International Semantic Search Workshop*. ACM, 2010, pp. 2:1–2:8.

[37] J. Pérez-Iglesias, J. R. Pérez-Agüera, V. Fresno, and Y. Z. Feinstein, "Integrating the Probabilistic Models BM25/BM25F into Lucene," *CoRR*, vol. abs/0911.5046, 2009. [Online]. Available: http://arxiv.org/abs/0911.5046

[38] G. J. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*, 2nd ed. Wiley-Interscience, March 2008.

[39] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006. [Online]. Available: http://books.google.it/books?id=Clc7PwAACAAJ

[40] L. Dolamic and J. Savoy, "When stopword lists make the difference," *J. Am. Soc. Inf. Sci. Technol.*, vol. 61, no. 1, pp. 200–203, Jan. 2010. [Online]. Available: http://dx.doi.org/10.1002/asi.v61:1

[41] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[42] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering - An Introduction*. Kluwer, 2000.

[43] V. Basili, G. Caldiera, and D. H. Rombach, *The Goal Question Metric Paradigm, Encyclopedia of Software Engineering*. John Wiley and Sons, 1994.

[44] J. K. Cullum and R. A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol. 1*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002.

[45] L. Mazzeo, A. Tolve, R. Branda, and G. Scanniello, "Linking e-mails and source code with lasco," in *Proc. of the European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 2013.

Using Function Point Analysis and COSMIC for Measuring the Functional Size of Real-Time and Embedded Software: a Comparison

Luigi Lavazza Sandro Morasca Davide Tosi
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
{luigi.lavazza, sandro.morasca, davide.tosi}@uninsubria.it

Abstract— **Function Points Analysis and the COSMIC method are very often used for measuring the functional size of programs. The COSMIC method was proposed to solve some shortcomings of Function Points, including not being well suited for representing the functionality of real-time and embedded software. However, little evidence exists to support the claim that COSMIC Function Points are better suited than traditional Function Points for the measurement of real-time and embedded applications. To help practitioner choose a method for measuring real-time or embedded software, some evidence of the merits and shortcomings of the two methods is needed. Accordingly, our goal is to compare how well the two methods can be used in the functional measurement of real-time and embedded systems. To this end, we applied both measurement methods to the situations that occur quite often in real-time and embedded software and are not considered by standard measurement practices. Our results indicate that, overall, COSMIC Function Points are better suited than traditional Function Points for measuring characteristic features of real-time and embedded systems.**

Keywords - Functional Size Measurement, Function Point Analysis, COSMIC Function Points, Real-time software, Embedded software.

I. INTRODUCTION

Several methods have been proposed to estimate the development effort of a software product, given the characteristics of the product itself and its development process. Software size plays a special role in effort estimation, as it is the main input used by the vast majority of effort estimation models. Accordingly, measures of *functional* size are used in early effort estimation models, since other measures –like Lines of Code– are not available in the early development phases. Functional measures quantify the functional size of a software application, as defined in the requirements specification documents. The need for software development estimation and size measurement applies to RT software as well [1].

The available functional sizing methods are evolutions of Function Points Analysis (FPA), originally proposed by Allan Albrecht [2]. The International Function Points User Group (IFPUG) maintains the definition of the method and publishes and regularly updates the official Function Point (FP) counting manual [3][4]. Effort estimation methods have been defined, and tools supporting them have been developed, which require the size in FP as the main input [5].

FP are generally not considered well suited for measuring the functional size of embedded applications. The reported motivation is that FP –conceived by Albrecht when the programs to be sized were mostly Electronic Data Processing applications– capture well the functional sizes of data storage and data movement operations, but are ill-suited for representing the complexity of control and elaboration that are typical of embedded and real-time software.

The COSMIC method was defined to overcome some limitations of FPA. The COSMIC method [6] redefines FPA's basic principles of functional size measurement in a way that applies equally well to traditional "business" application and other applications, including real-time and embedded ones. Specifically, the COSMIC method counts the data movements (entries, exits, reads, and writes) that involve data groups (corresponding approximately to FPA's logic files) in each functional process (corresponding to FPA's elementary processes). The result is a functional size measure called COSMIC Function Points (CFP).

Even though it is traditionally considered not well suited for real-time and embedded applications, FPA can be applied to embedded software via a careful interpretation of FP counting rules [7]. Moreover, it is known that many real-time projects have actually been measured using FPA. On the contrary, there is little *analytic* evidence of successful applications of the COSMIC method to real-time and embedded applications. This paper aims at providing some evidence about the suitability of FPA and the COSMIC method to measure real-time embedded software.

Both FPA and COSMIC methods require the representation of user requirements according to a method-specific model of software (e.g., the FP model includes logic files and elementary processes, while the COSMIC model includes functional processes and data movements). Measurement is then based on counting the elements of these models according to given rules. To measure real-time and embedded software, it is of critical importance that representative models can be correctly derived from the user requirements. To test this ability, we consider a somewhat extensive –though necessarily incomplete– set of typical and representative features of real-time embedded software and apply FPA and COSMIC to each of them. The comparison of the two methods provides useful indications to the developers that have to choose a functional size measurement method.

Even though both FPA and COSMIC methods aim at measuring the size of Functional User Requirements (FUR), there are a few reasons that suggest that the COSMIC method may be preferable. First, CFP are defined in a simple and sound way, while the definition of FP has been widely criticized, e.g., because the weighting mechanism make unclear whether FP are a measure of size or effort [8], or because the inherent subjectivity of FPA leads even certified measurers to provide different size measures for the same application [9][10]. Finally, the COSMIC method, which does not require a thorough analysis of data and allows for analyzing transactions at coarser granularity level, is somewhat faster and less expensive than FPA.

So, managers have a few reasons to prefer the COSMIC method over FPA. However, evidence concerning the suitability of the COSMIC method for measuring real-time software is still missing. This paper aims at filling this gap.

In this paper, we enhance the work reported in [1] by considering additional characteristics of real-time and embedded software –namely the usage of clocks and timers– that could make the application of functional size measurement rules challenging. Consequently, the discussion on the comparison of FP and CFP is extended to the newly considered cases. In addition, the section on related work was also extended.

This paper is of interest to researchers and practitioners who are familiar with the usage of Function Point Analysis and the COSMIC method for measuring traditional software applications. Accordingly, we take for granted that the readers are familiar with the concepts and terms of functional size measurement in general and FPA and the COSMIC method in particular. Readers who are not familiar with Functional Size Measurement methods can have a look at a concise introduction to FPA and COSMIC [29] and at a glossary of metrics terms [30].

Throughout the paper, we refer exclusively to Unadjusted Function Points (UFP) for FPA, because UFP are more commonly used than adjusted Function Points and because UFP are recognized as an ISO standard [4], while FP are not.

The paper is organized as follows: Section II accounts for related work. Section III presents a set of modeling and measurement problems that occur frequently in real-time and embedded software developments. In Section IV, FPA and COSMIC methods are applied to the cases illustrated in Section III, while Section V draws some conclusions and outlines future work.

II. RELATED WORK

There is a fairly large body of literature aimed at extending the scope of functional size measurement to software applications that do not belong to the Information Systems domain, for which FPA was originally conceived by Albrecht. An overview of these proposals (rather old but still relevant) can be found in [13]. Among the notable attempts to adapt FPA to real-time software are:

– Feature Points [14], which include an algorithmic element and define new environmental complexity factors.
– Mark II Function Points [15][16], which refine and extend the traditional function point transaction model and environmental factors.
– Asset-R [17], which extends the applicability of function points to real-time systems by considering issues like concurrence, synchronization, and reuse. It also accounts for architectural, language expansion, and technology factors to generate the size estimate.
– 3D Function Points [18], which consider three dimensions of the application to be measured: Data, Function, and Control. The Function measurement considers the complexity of algorithms; and the Control portion measures the number of major state transitions within the application.
– Application Features [19], which aims at the early estimation of the size of application in the process control domain.
– Counting practices for highly constrained systems [20], which address issues such as boundary identification and internal processing.

Also the IFPUG published a Case Study that shows how to apply FPA to real-time software [21].

Another set of proposed approaches to make FP measurement applicable to real-time software took into account the object-oriented programming paradigm. Actually, these approaches address every type of object-oriented program or model, including real-time and control applications.

Object Points [22] are an object-oriented approach that measures the external, internal, application and object size of object-oriented systems. Objects are viewed as mini applications where each object encapsulates data and operations. A simple mapping is established between object operations (services) to transactions, and object data (attributes) to ILFs.

Class points [23] are based on the number of services required (NSR), the number of external methods (NEM) and the number of attributes (NOA) of classes. The complexity of a class is evaluated on the basis of its NSR, NEM and NOA, and then classes are weighed according to their complexity and type. Class types are: Problem Domain, Human Interaction, Data Management, Task Management. The sum of the weights gives the number of class points.

Object-oriented FPs [24] are computed following the function point counting procedure. Classes within the application boundary correspond to ILFs, while classes outside the application boundary (including libraries) correspond to EIFs. Inputs, Outputs and Inquiries are all treated in the same way: they are called generically "service requests" and correspond to class methods. The complexity of ILFs and EIFs depends on the number and type of attributes and associations. The complexity of service requests depends on the number and type of method parameters. Several ways of considering class aggregates and generalization hierarchies are proposed, thus the measured size depends on the criterion used to consider class aggregation and generalization.

Among the proposed approaches –none of which seems to have succeeded in gaining market acceptance– Full

Function Points (FFP) are quite relevant. FFP [25] take into account the differences between traditional applications and real-time applications by extending the FPA by means of new data and transactional function types:

– Updated control group: a group of data –used by the application to control, directly or indirectly, the behavior of an application or of a mechanical device– updated by the application being measured.
– Read-only control group: A group of control data used, but not updated, by the application being counted.
– Entry / Exit: A sub-process that receives / sends control data across the application's boundary.
– Read / Write: A sub-process that reads / writes a group of control data.

Also FFP did not prove successful in dealing with the functional size measurement of real-time software; hence, their authors decided to thoroughly review their definition, thus arriving at the definition of COSMIC function points [6][26]. CFP retain the basic principles of functional size measurement as in FPA, but they are defined in a manner that applies equally well to traditional "business" application and to other applications, including the real-time and embedded ones.

Several papers have been written on the suitability of Full Function Point and COSMIC Function Points to measure real-time software.

Desharnais and Morris stress the possibility of identifying and measuring different layers with the COSMIC method, and dealing with the "cut-off" effect we discussed in Section IV. IV.D [27].

Oligny et al. report about the applicability of FFP in general, based on the experiences gained while measuring seven projects (four of which real-time) [28]. The discussion does not address any of the details reported in our paper.

In conclusion, the literature is relatively rich in proposals for extending or adapting existing functional size measurement methods to real-time and embedded software; however, none of such proposals appears to be widely used in practice (possibly with the partial exception of Mark II Function Points [15], which were also standardized [16]).

So, the popularity of FPA and COSMIC suggested that they are candidates for real-time and embedded software measurement. However, nobody –to the best of our knowledge– investigated the actual applicability of IFPUG and COSMIC measurement rules to real-time and embedded software.

III. CASE STUDIES FOR FUNCTIONAL SIZE MEASUREMENT OF REAL-TIME EMBEDDED SOFTWARE

Here, we illustrate a set of typical features of real-time and embedded software that are difficult to represent by means of the models that underlie the definition of functional size measurement methods. All of the cases shown here are derived from the first author's experience gained in measuring seven avionics applications in a large European company. So, the proposed set of cases is of empirical origin: during the measurement, the cases presented here emerged as those particularly challenging for functional size measurement. Even though the cases considered here were all derived from the avionic domain, they were observed in quite different applications. Accordingly, we believe that the cases presented here are representative of the challenging cases that can occur when measuring real-time and embedded software applications.

Most examples are illustrated by means of sequence diagrams, according to the measurement-oriented modeling methodology proposed in [11] and used in [12]. It is assumed that the reader is familiar with UML.

A. *Embedded processes having multiple purposes*

In embedded software, several processes often include both updating some data and producing some result. Consider for instance a process that initializes and tests a piece of hardware (Fig. 1). The initialization and test of several hardware devices are performed by means of a single command: the initialization command is sent to the devices and the resulting state is sent back, so that it is possible to check whether the device is working correctly.

In these cases, the initialization and the test are both necessary and equally important.

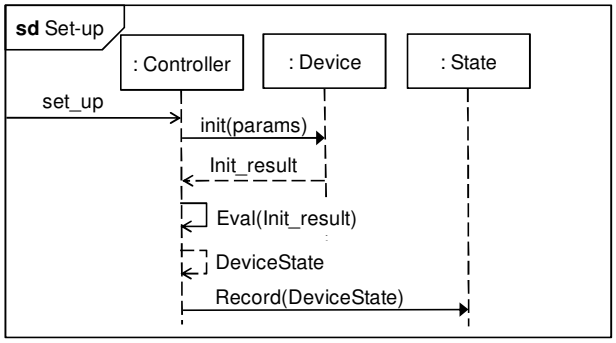


Figure 1. Inizialization of devices: the "main purpose" is not evident.

B. *Transactions defined at very low level*

Requirements often concern very low level operations, thus making it difficult to identify functions that match the definition of Base Functional Components.

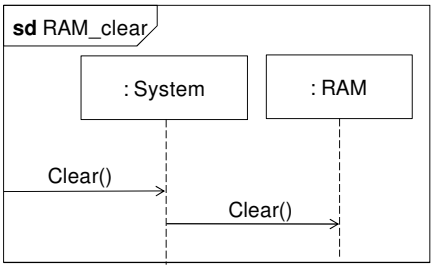


Figure 2. RAM clearing process.

1) *Memory vs. data*

In embedded software, the use of RAM as a whole introduces new requirements. For example, a piece of software embedded on board of a military airplane should clear the whole RAM under given circumstances, e.g., if the

airplane crashes in an enemy zone (because the information stored in memory must not be made available to enemies). This requirement (Fig. 2) is peculiar in that it is about the whole RAM, not about some specific user-relevant piece of data.

2) Memory mapped I/O

In embedded systems, updating a variable and sending data to a device can be extremely similar operations. For instance, when I/O is memory-mapped, both mentioned operations write registers or RAM locations (Fig. 3).

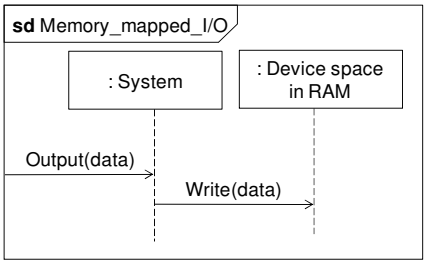


Figure 3. Memory-mapped I/O.

3) Processes that do not terminate properly

In embedded software, it is often required that a function terminate by jumping to a given location. This situation is illustrated in Fig. 4: the initialization function terminates by executing the set-up function (described in Fig. 17).

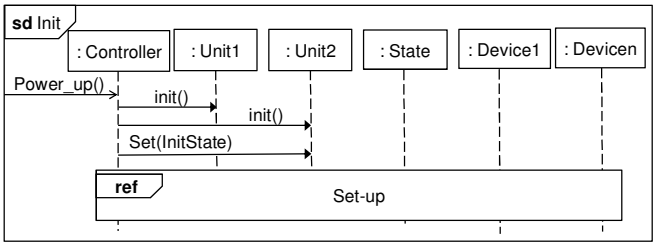


Figure 4. A function that ends with a jump to another function.

C. Taking into account the devices

In traditional software applications, functions are usually invoked by the user and end by either updating some internal data, or outputting some information. In embedded applications, the situation can be very different. Often it is some hardware device (not a user) that acts as both the cause that determines the execution of the function and the destination of the produced data or signals.

For instance, functional processes are often initiated by clocks and timers.

1) The Clock

Some functions are triggered periodically by clock signals. In such cases, the clock acts as a user that invokes a function: in fact, the resulting behavior is the same obtained by a human user that periodically invokes the function.

This situation is illustrated in Fig. 5: in this example, an aircraft is equipped with sensors, which collect navigation data, and a clock periodically invokes a sensor manager that asks navigation data from the sensors and sends the returned data to the flight control unit.

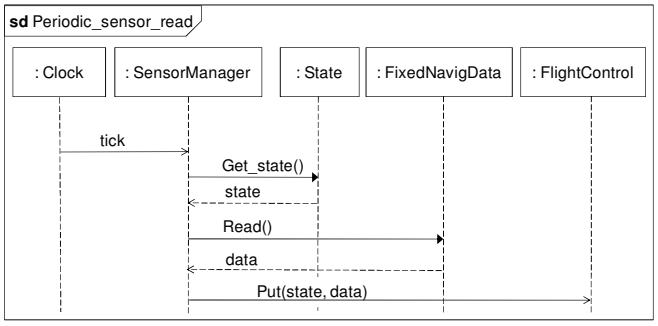


Figure 5. An elementary function triggered by the clock.

2) Timers

In embedded systems, functions can be triggered by timers, or they can terminate after programming a timer. Consider the following specifications:

Spec. A (Fig. 6): "The program sends a request for data to device X, then reads the data sent by X and stores them for later use."

Spec. B (Fig. 7): "The program sends a request for data to device X, waits for 10 ms, then reads the data from X and stores them for later use."

Spec. C (Fig. 8): "The program sends a request for data to device X; then, every 10 ms the program checks whether the data from X are ready: if so it stores them for later use."

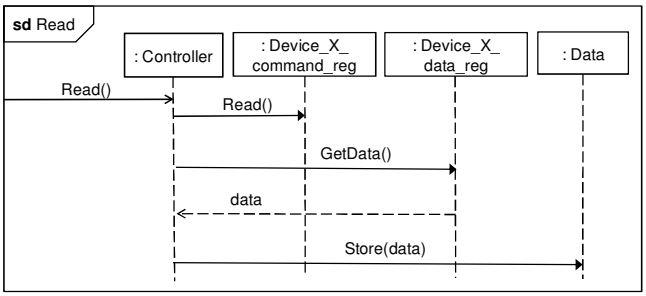


Figure 6. Device read specifications not mentioning time.

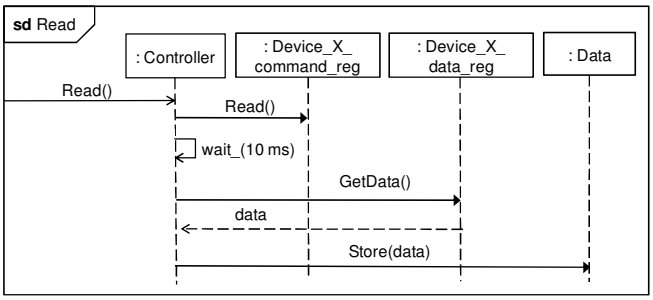


Figure 7. Device read specifications not mentioning time delay.

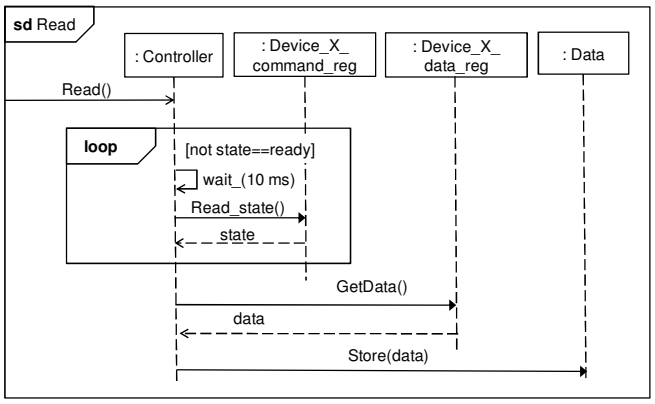


Figure 8. Device polling specifications.

All of the above specifications could be rewritten by explicitly mentioning the use of timers. For instance, Spec. C could be rewritten as follows:

Spec. D: "The program sends a request for data to device X and programs a 10 ms timeout; upon receiving the timeout signal, the program checks whether the data from X are ready: if so it stores them for later use and disables the timer, otherwise, a new 10 ms timeout is programmed."

The first part of this specification is described in Fig. 9, while the second part is described in Fig. 10

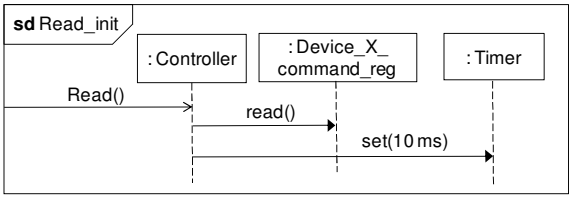


Figure 9. A transaction that programs a timer.

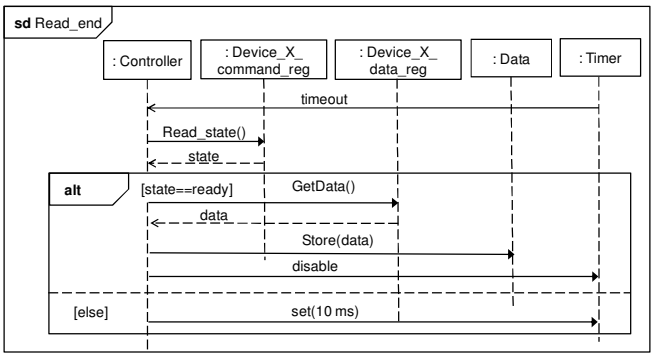


Figure 10. The timer triggers the conclusion of the operation.

3) Considering the role of the Operating System in I/O

Let us consider the following requirements for an I/O functionality (described in Fig. 11): "upon request by the controller, data are retrieved from an I/O channel, according to the criteria stored in the I/O channel table. When all the data have been read, they are suitably converted and sent back to the controller." It is often the case that the I/O operation has to be carried out with the help of the Operating System and the requirements can be implemented by means of two functions, illustrated in Figs. 12 and 13. The first function (Fig. 12) is invoked by the controller and prepares an I/O request for the OS and a subsequent system call. The second function (Fig. 13) is triggered by the interrupt from the I/O device and involves reading the data from the channel, elaborating them, and sending them back to the controller. The execution of this "function" is done partly by the OS (by a driver that will have to be implemented as a part of the application development) and partly in the section of the application devoted to I/O.

If the development also includes the construction of a driver for the considered I/O device, taking into account the size of the corresponding code will contribute to produce a more accurate effort estimate. In other words, it appears reasonable to count two functions, corresponding to the "elementary processes" described in Figs. 12 and 13.

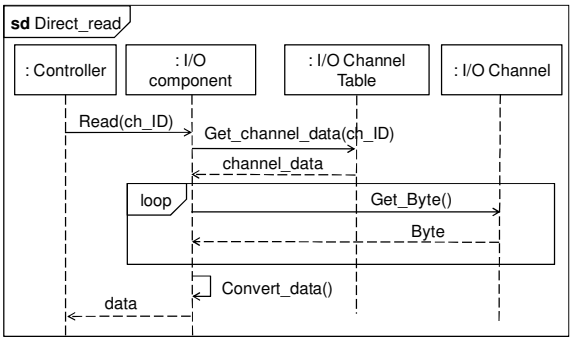


Figure 11. Process featuring direct access to I/O channels.

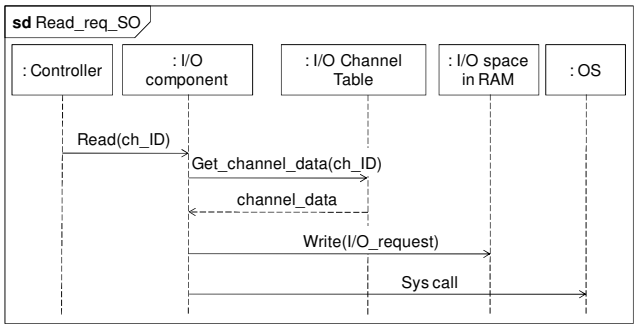


Figure 12. Process Access to I/O channels via the O.S.

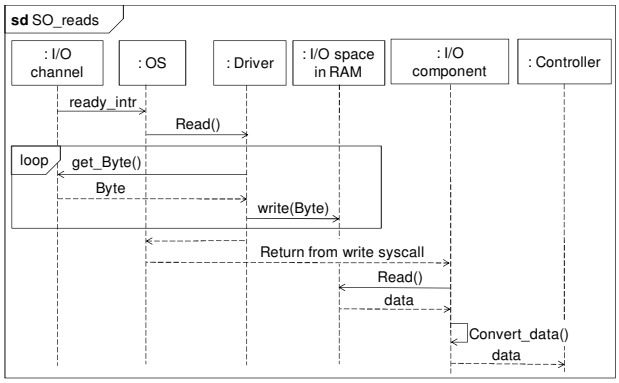


Figure 13. The O.S. handles the I/O.

4) Multi cycle operations

In real-time systems, it is not unusual that a function is too long to fit into one execution cycle. In such cases, it is rather common to split the function into two (or more) pieces that are executed in consecutive execution cycles. Here are two typical examples:

- The function transfers data via a buffer. The data to be transferred do not fit in the buffer. The transfer is split into n cycles: in each cycle 1/n of the data are copied into the buffer.
- The function, triggered by the tick, takes a time longer than the cycle duration (i.e., the time between two consecutive ticks) to execute. Thus, the transfer is split into multiple consecutive cycles.

An example is given in Figs. 14 and 15: an output operation is split over two consecutive clock cycles. In the first cycle (Fig. 14), the application outputs the data from Data_1 and sets the State to represent that there is a pending output operation. In the following cycle (Fig. 15), the State indicates that the output operation must be completed, thus data are read from Data_2 and sent to the output device.

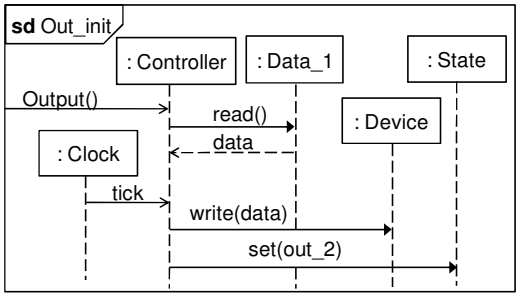


Figure 14. Output: first cycle.

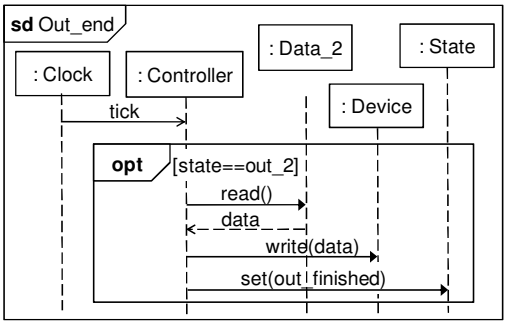


Figure 15. Output: second (final) cycle.

These cases are often described in the requirements, since they deal with the real-time behavior of the application, which is typically explicitly accounted for in the requirements specification. However, requirements specifications could not state explicitly that the function should be split, i.e., requirements could just describe the whole operation as in Fig. 16.

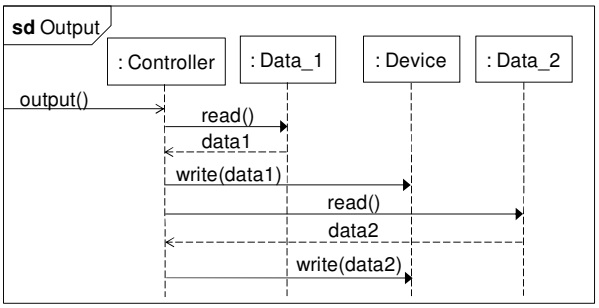


Figure 16. Output, not split.

D. Long processes

In embedded software, functions are often "service routines" that perform rather long tasks; e.g., the requirements specify that "the connected devices are tested, and the result (a 'pass' value or the set of diagnostics) is sent to the controller, which stores it for later use." Fig. 17 illustrates the situation with 4 different device types.

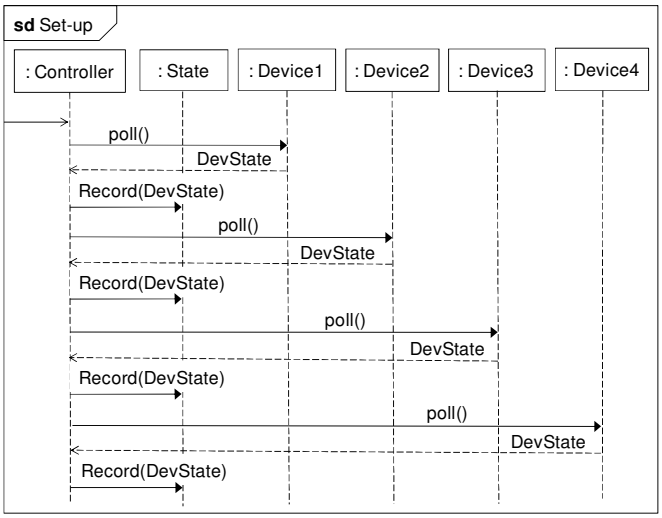


Figure 17. A long transaction.

E. Unusual data

Embedded applications often include constant data structures (e.g., data mapping tables or bit masks) that require a non-negligible design effort, which we would like to take into account. An example is shown in Fig. 12: for each request to read an I/O channel, the I/O component reads from the channel table how many bytes must be read from the channel and how they should be interpreted. The channel table is a read-only structure that describes how to manage the I/O channels.

With respect to other elements of the system, the channel table differs only in that it is read-only; apart from that, it concerns information that is relevant to the user and it must be properly designed to be effectively and efficiently read.

F. Complex elaborations

In real-time and embedded applications, some operations can be complex. Consider for instance the generic flight control operations described in Fig. 18: it is quite likely that the computation of the flight control data is rather complex.

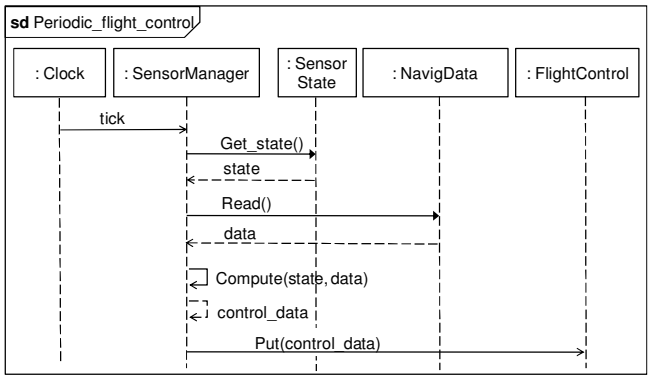


Figure 18. Sensor-driven flight control.

IV. APPLYING FPA AND COSMIC TO REAL-TIME EMBEDDED SOFTWARE

This section illustrates the application of FPA and COSMIC methods to the cases described in Section III.

A. Embedded processes having multiple purposes

According to the IFPUG counting rules [3][4], the size of a function varies according to its type (external input, output or query). The type is determined by the "main purpose" of the function, according to the requirements. However, it may be difficult to decide what the main purpose is, since both the external input and the external output can update internal data and report a result, as in our case. In conclusion, measures based on FPA have some degree of subjectivity that can be hardly avoided.

The problem described above does not apply to COSMIC measurement, since all processes are treated in the same way, regardless of their purpose.

B. Transactions defined at very low level

1) Memory vs. data

According to the principles of FPA, in a case like the one described in Section III.B.1) one should count the memory clearing function as an external input. In that case, since every External Input (EI) manages an Internal Logical File (ILF), we should consider the RAM an ILF. On the one hand, counting the RAM as an ILF does not appear correct with respect to the rules, since logic data files should represent a homogeneous set of related data (which RAM is not), on the other hand, not considering the RAM as an ILF is an inconsistency, as all EI have to deal with an ILF.

There is a similar problem with the COSMIC method, as the process writes in the RAM: accordingly, we should consider a write data movement. However, this implies that the RAM is classified as a data group, which does not appear perfectly coherent with the COSMIC rules.

In the COSMIC method, there is no rule that clearly prevents treating the RAM as the object of interest involved in the process that clears the RAM. Accordingly, we would have a process involving a write data movement.

2) Memory mapped I/O

When I/O is memory-mapped, an output operation can be modeled as an External Output (EO) in FPA but also as an EI, since the output is obtained by writing registers or RAM locations (see Fig. 3). The choice affects the resulting measure, since EI and EO have different weights. With the COSMIC method, you still can model the operation as a Write or an Exit data movement, but the choice does not affect the final measure, since every data movement contributes exactly one CFP.

3) Processes that do not finish properly

According to FPA, a transaction function has to be self-contained and leave the application being counted in a consistent state. In embedded software, it is often required that a function terminates by jumping to a given location (Fig. 4). In this case, the transaction is not self-contained and does not leave the program in a consistent state. FPA does not suggest how to deal with this type of functions. Just ignoring them would not be a good idea, since it takes some effort to implement these functions; hence, we want them to contribute to the functional size of the application. Actually, there is no other way of dealing with these cases than just ignoring the constraints imposed by the IFPUG and counting the functions, considering their behavior down to the final jump. The same problem occurs when the COSMIC method is used, since functional processes are defined as FPA transactions, in essence.

C. Taking into account the devices

1) The Clock

In functions that are triggered periodically by clock signals (as in Fig, 5), the clock acts as the user that invokes the function.

From a COSMIC point-of-view, the clock is the functional user that generates the triggering Entry (i.e., a message that informs the software that the functional user is initiating a functional process). The possibility that a device acts as a functional user is explicitly stated in the COSMIC counting manual [6]. Accordingly, the functional process illustrated in Fig. 5 involves the following data movements:

- The entry of the clock tick;
- Reading the current state;
- The request to for navigation data (an exit);
- The entry of navigation data;
- The output of state;
- The output of navigation data.

The sensor is another functional user. For this software, an event occurs when it is time to update the navigation data: the clock triggers the software, by sending it a message (triggering Entry).

The considerations reported above can be applied to FPA as well. Thus, we shall simply consider the clock as a user that can originate the execution of a function. The rest of the measurement is carried out easily according to FPA rules [3].

2) Timers

Let us consider Spec. A (Fig. 6): in this specification there is no mention of the time delay between the request to read and the retrieval of data. According to FPA, the specification involves a single transaction (an external input, if the main intent is storing the data retrieved from the sensor). Similarly, it is a single COSMIC functional process.

Specifications B (Fig. 7) and C (Fig. 8) are functionally equivalent to Spec. A, with the only difference that they mention time. By the way, Spec. B and C are also similar to each other, the only difference being that, in Spec. B, we are sure that, 10 ms after issuing the read command, the required data are ready, while in Spec. C this is not true. However, as with Spec. A, we have just one transition according to FPA, or a functional process according to the COSMIC method.

Specifications B and C (and possibly A as well) can be implemented with or without using a timer: in fact the 10 ms waiting could be achieved via a sort of busy loop. Of course, the functional size (either in FP or in CFP) of the specifications does not depend on how it will be implemented.

The problem is that the same operation described by Specifications B and C could be described as in Spec. D (Fig. 9 and Fig. 10). In this case, the analyst is just specifying delays by means of timers, which –by the way– are the most obvious means to implement the operation. As a consequence, we have two FPA transactions: the one that initialized the device (described in Fig. 9) and the one that reacts to timeout signals (Fig. 10).

In practice, specifications B and C are both low complexity External Input transactions (under the hypothesis that the data read from the sensor includes a small set of DET), accounting for 3 FP.

From a COSMIC point-of-view, Spec. B involves 5 data movements (the Entry of the triggering event, the Exit of the Read command, the Exit of the GetData command, the Entry of data, the Write of data), while Spec. C involves 6 data movements (the Entry of the triggering event, the Exit of the Read_state command, the Entry of the state, the Exit of the GetData command, the Entry of the data, the Write of data).

Spec. D involves a low complexity External Output transaction (Fig. 9) and a low complexity External Input transaction (Fig. 10): therefore, it has a size of 3+4=7 FP. From a COSMIC point-of-view, Spec. B involves a functional processes (Fig. 9) comprising 3 data movements, and a functional process (Fig. 10) comprising 7 data movements: therefore, it has a size of 3+7=10 CFP.

In conclusion, the problem here is that mentioning the timers in the specifications (which is quite natural for real-time software analysts) causes the functional size of the specified operations to increase substantially.

3) Considering the role of the Operating System in I/O

With both FPA and COSMIC methods, the measurement of the process represented in Fig. 11 is quite straightforward. The problem here occurs when the development must also include the construction of a driver for the considered I/O device, since taking into account the size of the corresponding code will contribute to producing a more accurate effort estimate. In other words, it appears reasonable to count two functions, described in Figs. 12 and 13.

This requires a deviation from the FPA counting practice, since FPA does not take into account the existence of different "layers": with FPA you can only measure requirements at the single abstraction level corresponding to the user's point of view, and the user is not aware of the OS and what happens in the OS.

With the COSMIC method, it is possible to explicitly model and measure the layers that compose the software application. The sum of the sizes of the layers is generally greater than the size of the whole application corresponding to the point of view of the user (who is not aware of the existence of layers). So, the measure of layers is exactly what is needed to take into account the size of the OS parts that are being developed.

4) Multi cycle operations

The cases described in Section III.C.4) suggest that the value of a functional size measure can depend on how requirements are written. Let us consider the case when requirements specifications do not state explicitly that the function should be split (Fig. 16): if Data_1 and Data_2 account for 10 DET each, the transaction is a high complexity EO (having 3 FTR and 21 DET), whose size is 7 FP. When requirements specifications prescribe that the function be split (Fig. 14 and 15) we have two average complexity EO (3 FTR and around 12 DET each), whose size is 10 FP in total. When requirements specifications do not state explicitly that the function should be split, the COSMIC method identifies one functional process sized 5 CFP, since it involves 5 data movements (the Entry, the Reads of Data_1 and Data_2, and the corresponding Exits). When requirements specifications prescribe that the function be split, according to the COSMIC rules we have two functional processes, one involving 5 data movements (the Entry that triggers the operation, the Read of Data_1, the Entry of the clock tick, the Exit to the device, the Write of the state), and one involving 4 data movements (the Entry of the tick, the Read of Data_2, the Exit to the device, the Write of the state); the total size is thus 9 CFP.

In conclusion, both methods provide measures of size that depend on how requirements are written. This is a characteristic of the methods that has to be taken into account, as it affects the resulting measures.

D. Long processes

A well known problem with Function Points is the so-called "cut-off" effect: a function cannot contribute more than 7 FP to the functional size, regardless how many DETs it moves and how many FTRs it involves. This is a relevant problem, especially in embedded software, where functions are often "service routines" that perform rather long tasks, like in the example illustrated in Section III.D and Fig. 17.

Fig. 17 illustrates the situation with 4 different device types. According to the IFPUG counting rules, this is a single

transaction. If the device states contain on average 5 (or more) parameters, then the transaction is a complex one. The problem here is that if we had 5 or more different types of devices, the number of FP would not increase with the number of devices: according to FPA, we would have just one complex EI. This is a problem, because in practice the development effort increases with the number of device types, since each device type provides different status data, which need to be interpreted in a specific way.

FPA hides from the estimation methods how much bigger a function is (thus more expensive to build) than another that classifies as complex. The COSMIC method, on the contrary, does not suffer from the cut-off effect. In a case like the one in Section III.D and Fig. 17, the size in CFP takes into account *all* the data movement, whose number is proportional to the number of devices.

E. Unusual data

According to FPA, data functions are either internal data "maintained" (i.e., modified) by the application, or external data (maintained outside the application). Constant data are treated as "decoding data" and explicitly excluded from the counting [3]. However, it seems that the authors of the IFPUG manual had in mind simple "zero effort" constants when they wrote the rules concerning the constant data.

To account for the fact that a constant data structure will require some design effort, it is necessary to deviate from the IFPUG rules, and count a "constant ILF." For instance, in the example illustrated in Fig. 12, one should count an ILF for the channel table; consistently, a FTR for each access to the table should be considered.

The COSMIC method does not count data directly; that is, no fraction of the size measures accounts for data. On the contrary, data movements are counted without considering whether the data being moved are constant or not. In conclusion, this case does not pose any additional difficulty to the application of the COSMIC method.

F. Complex elaborations

Both FPA and COSMIC methods base the measurement of size on the number of processes and the amount of data handled. For instance, the process described in Fig. 18 is considered as an EO (with a maximum size of 7 FP) or a functional process accounting for 4 CFP (as it involves 4 data movements). None of the two methods considers the complexity of the computations performed: the fact that the "Compute" operation performed in the process is simple or complex does not change the size of the process.

This is clearly a shortcoming of the two methods, since the development effort is very likely proportional to the complexity of the functions to be implemented.

V. CONCLUSIONS

The results of our analysis (summarized in Table I) show that some situations that are typical of real-time and embedded applications make it necessary to interpret or "bend" the rules provided by official measurement manuals [3][6]. However, this happens more often for IFPUG Function Point Analysis than with the COSMIC method.

Also the resulting measures are easily affected by the measurement choices made in FPA, while there are just a few cases (namely, processes terminating with a jump, multi-cycle operations, explicitly mentioned timers and complex elaborations) that can affect the measures in CFP.

TABLE I. COMPARISON OF FSM METHODS

Case	FPA		COSMIC	
	Easy application of rules	*Measure affected*	*Easy application of rules*	*Measure affected.*
Multiple purpose processes	×[a]	×	✓	✓
Memory data	×	×	✓	✓
Memory mapped I/O	×	×	×	✓
Processes terminating with jump	×	×	×	×
Clock	✓	✓	✓	✓
Timers	✓	×[b]	✓	×[b]
OS involved in I/O	×	×	✓	✓
Multi cycle operations	✓	×[b]	✓	×[b]
Long processes	×	×	✓	✓
Unusual data	×	×	✓	✓
Complex elaborations	×[c]	×	×[c]	×

[a] The application of the rule is subjective.

[b] The measures depend on how requirements are written.

[c] Elaboration complexity is just not accounted for by any rule.

In conclusion, the original claims that the COSMIC method is more suitable than FPA for measuring real-time and embedded applications appear justified. The cases that were used to evaluate the applicability of FPA and COSMIC to real-time and embedded software are sort of application-independent patterns: accordingly, the results reported here are expected to be applicable to a wide range of real-time and embedded software applications.

A straightforward consequence of the study reported here is that the COSMIC method is more suited for the functional measurement of real-time software; however, considering that functional size measures are often used for effort estimation, a problem could arise with the availability of CFP-based models of real-time software development effort. To derive such models, historical data are needed. The study reported here could also provide hints for converting FP measures into CFP measures, thus obtaining the datasets that are necessary to derive effort models.

In any case, neither FPA nor the COSMIC method account for the complexity of the required elaboration. This may be a problem in the real-time embedded context, since some processes can be really very complex and require a relevant amount of development effort.

Future work involves assessing measures that represent not only the functional size of real-time applications as done by FPA and COSMIC methods, but can represent also the complexity of the required elaboration.

ACKNOWLEDGMENT

The work reported here was supported by the FP7 Collaborative Project S-CASE (Grant Agreement No 610717), funded by the European Commission and by project "Metodi, tecniche e strumenti per l'analisi, l'implementazione e la valutazione di sistemi software," funded by the Università degli Studi dell'Insubria.

REFERENCES

[1] L. Lavazza and S. Morasca, "Measuring the Functional Size of Real-Time and Embedded Software: a Comparison of Function Point Analysis and COSMIC", 8th Int. Conf. on Software Engineering Advances – ICSEA 2013, October 27 - November 1, 2013 - Venice, Italy

[2] A.J. Albrecht, Measuring Application Development Productivity, Joint SHARE/ GUIDE/IBM Application Development Symposium, 1979, pp. 83-92.

[3] International Function Point Users Group. Function Point Counting Practices Manual - Release 4.3.1, January 2010.

[4] ISO/IEC 20926: 2003, Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting Practices Manual, Geneva: ISO, 2003.

[5] J. E. Matson, B. E. Barrett, and J M. Mellichamp, "Software development cost estimation using function points,", IEEE Transactions on Software Engineering, vol.20, no.4, Apr 1994, pp.275-287.

[6] COSMIC – Common Software Measurement International Consortium, The COSMIC Functional Size Measurement Method - version 3.0.1 Measurement Manual, May 2009.

[7] L. Lavazza and C. Garavaglia, "Using Function Points to Measure and Estimate Real-Time and Embedded Software: Experiences and Guidelines", ESEM 2009, Lake Buena Vista, FL, USA, October 15-16, 2009, IEEE, pp. 100-110.

[8] A. Abran and P.N. Robillard "Function points: a study of their measurement processes and scale transformations", Journal of Systems and Software, vol.25,n.2, Elsevier, 1994, pp.171-184.

[9] C. Kemerer, "Reliability of Function Points Measurement: a Field Experiment," Comm. ACM, Vol. 36, No. 2, 1993, pp. 85-97.

[10] J.R. Jeffery, G.C. Low, and M.A Barnes, "Comparison of Function Point Counting Techniques," IEEE Trans. Software Eng., Vol. 19, No. 5, 1993, pp. 529-532.

[11] L. Lavazza, V. del Bianco, and C. Garavaglia, "Model-based Functional Size Measurement", 2nd Int. Symp. on Empirical Software Engineering and Measurement – ESEM 2008, Kaiserslautern, Germany. October 9-10, 2008, pp. 100-109.

[12] L. Lavazza and V. del Bianco, "A Case Study in COSMIC Functional Size Measurement: the Rice Cooker Revisited", IWSM 2009, Amsterdam, November 2009, pp. 101-121.

[13] T. Hastings, "Adapting Function Points to contemporary software systems: A review of proposals", 2nd Australian Conference on Software Metrics. Australian Software Metrics Association, 1995.

[14] C. Jones, Applied Software Measurement - Assuring Productivity and Quality, McGraw-Hill, New York, 1991.

[15] C.R. Symons, "Function Point Analysis: Difficulties and Improvements", IEEE Transactions on Software Engineering, Vol. 14, No. 1, January, 1988, pp. 2-11.

[16] ISO/IEC 20968: 2002, Software engineering Mk II Function Point Analysis. Counting Practices Manual, International Standardization Organization, ISO, Genève, 2002.

[17] D. J. Reifer, "Asset-R: A Function Point Sizing Tool for Scientific and Real-Time Systems", Journal of Systems and Software, Vol. 11, No. 3, March 1990, pp. 159-171.

[18] S. A. Whitmire, "An Introduction to 3D Function Points", Software Development, Vol. 3 No.4, 1995.

[19] T. Mukhopadhyay and S. Kekre, "Software Effort Models for Early Estimation of Process Control Applications", IEEE Transactions on Software Engineering, Vol. 18, No. 10, October 1992, pp. 915-924.

[20] European Function Point Users Group, Function Point Counting Practices for Highly Constrained Systems, 1993.

[21] IFPUG, Case Study 4: Counts Function Points for a Traffic Control System with Real Time Components, International Function Point Users Group – IFPUG.

[22] S. A. Whitmire, "Applying Function Points to Object Oriented Software Models", in Software Engineering Productivity Handbook, J. Keyes Ed., New York, Windcrest/McGraw-Hill, 1993.

[23] G. Costagliola, F. Ferrucci, G. Tortora, and G. Vitiello, "Class Point: An Approach for the Size Estimation of Object-Oriented Systems", IEEE Transactions On Software Engineering, Vol. 31, No. 1, pp. 52-74, January 2005.

[24] G. Antoniol, C. Lokan, G. Caldiera, and R. Fiutem, "A Function Point-Like Measure for Object-Oriented Software", Empirical Software Engineering, 4 (3), September 1999.

[25] M. Maya, A. Abran, S. Oligny, D. St-Pierre, and J.-M. Desharnais, "Measuring the Functional Size of Real-Time Software" 9th European Software Control and Metrics Conference and 5th Conference for the European Network of Clubs for Reliability and Safety of Software (ESCOM-ENCRESS-98), Rome, Italy, 1998.

[26] ISO/IEC 19761:2003, Software engineering – COSMIC-FFP – A functional size measurement method, Geneva: ISO, 2003.

[27] J.M. Desharnais, P. Morris, "Measuring ALL the Software not just what the Business", IFPUG Conference, 1998.

[28] S. Oligny, J.M. Desharnais, A. Abran, "A Method for Measuring the Functional Size of Embedded Software", 3rd Int. Conf. on Industrial Automation, pp. 7-9, 1999.

[29] L. Lavazza, V. del Bianco, and Geng Liu. "Analytical convertibility of functional size measures: a tool-based approach."" Joint Conference of the 22nd Int. Workshop on Software Measurement and the 7th Int. Conf. on Software Process and Product Measurement. IEEE Computer Society, 2012.

[30] Total Metrics. Glossary of metrics terms. at http://www.totalmetrics.com/resources/software-metrics-glossary (accessed on May 16th, 2014).

Confirming Design Guidelines for Evolvable Business Processes Based on the Concept of Entropy

Peter De Bruyn, Dieter Van Nuffel, Philip Huysmans, and Herwig Mannaert

Normalized Systems Institute (NSI)
Department of Management Information Systems
University of Antwerp
Antwerp, Belgium
{peter.debruyn, dieter.vannuffel, philip.huysmans, herwig.mannaert}@uantwerp.be

***Abstract*—Contemporary organizations need to be agile at both their IT systems and organizational structures (such as business processes). Normalized Systems theory has recently proposed an approach to build evolvable IT systems, based on the systems theoretic concept of stability. However, its applicability to the organizational level, including business processes, has proven to be relevant in the past and resulted among others in a set of 25 guidelines for designing business processes. In subsequent work, the Normalized Systems theory was confirmed and extended based on the concept of entropy from thermodynamics. This perspective allows for the investigation of the observability of IT systems or —at the organizational level— business processes. Therefore, this paper explores whether the guidelines which have been proposed to design business processes from an evolvability point of view can be confirmed or extended from the entropy reasoning as well. More specifically, the validity of 25 business process design guidelines is investigated for this purpose. While 9 of these guidelines were already analyzed in earlier work, this paper supplements the earlier analysis by including a discussion of the other 16 guidelines. Our results indicate that the investigated guidelines are rather consistent among both approaches: guidelines required to attain evolvability also enable low entropy (i.e., high observability) and vice versa. Part of one guideline was found to be not strictly necessary from the entropy viewpoint. Moreover, several guidelines were able to be refined to some extent based on our entropy reasoning or were subject to some additional nuancing.**

Keywords–Business Processes; Observability; Entropy; Stability; Normalized Systems

I. INTRODUCTION

This is a revised and extended version of a paper which was presented at The Eighth International Conference on Software Engineering Advances (ICSEA) and published in its corresponding proceedings [1].

Lack of organizational agility is often attributed to a lack of IT agility [2] as IT systems ensure the support or even automation of business processes. Consequently, organizational changes need to be reflected in both the business processes and their supporting information systems. This means that, instead of focusing solely on IT systems, attention for the design and agility of the business processes is needed as well. The explicit attention for the design of business processes emerged when the implicit work practices were automated using ERP systems [3]. It was recognized that the hard coding of the business processes in software packages resulted in a lack of adaptability of the processes [4]. As a result, the design of business processes gained a central role in organizations, separated from the design of information systems [3]. However, integration of business processes and information systems still needs to be achieved, and agility (or "evolvability") needs to be ensured on both levels.

Normalized Systems (NS) theory offers a theoretically founded way to design software systems which exhibit evolvability based on the systems theory's concept of stability, by proposing a limited set of design theorems [5], [6]. Applying the theory's rationale to the business process level has been shown feasible and resulted among others in a set of 25 guidelines for designing evolvable business processes, more specifically on the delineation of business processes and their constituting tasks [7]. In subsequent work, NS theory was confirmed and extended based on the concept of entropy from thermodynamics [8]. Such perspective enables the design of software systems having a high degree of observability (i.e., internal problems within the system are more easily detectable and traceable to the task generating this problem). At the software level, this extension resulted in additional theorems, while confirming the existing theorems. Moreover, similar entropy definitions were able to be defined at the business process level as well [9], [10], [11], [12]. Therefore, it is interesting to verify whether the guidelines which have been proposed for business processes (in order to make them more evolvable) can be confirmed or extended from the entropy reasoning as well. This paper explores this research area by applying the entropy reasoning to the considered set of business process guidelines. While 9 of these guidelines were already analyzed earlier work [1], this paper supplements the earlier analysis by including analysis of the remaining 16 guidelines.

Our paper will be structured as follows. First, we provide some theoretical background on Normalized Systems theory, its stability and entropy perspective, and their respective applications at the business process level (Section II). Afterwards, the analysis of the guidelines of Van Nuffel from an entropy perspective is presented in Section III. A discussion and our our conclusions are offered in Section IV and Section V, respectively.

II. THEORETICAL BACKGROUND

NS was introduced as a theoretically founded way for deterministically designing software architectures exhibiting a proven amount of evolvability. For this end, the systems theoretic concept of stability is applied [5], [6]. This implies that a bounded input function (e.g., "add data attribute") should result in bounded output values, even as time $T \rightarrow \infty$. Stated otherwise, this means that the required implementation effort for a particular change is only dependent on the nature of that change itself and not on the size of the system. It has been proven that at least four theorems (i.e., separation of concerns, data version transparency, action version transparency and separation of states) should be consistently applied in order to obtain such evolvable software architecture [5], [6]. Violations against these theorems can be observed at design time [6].

Later on, the theory has been proven to be applicable to the design of evolvable business processes [7]. Here, business processes are considered at their most elementary level (i.e., the "elementary tasks and elementary sequencing and design of these tasks" performed on information objects). To obtain stability, it is required that changes to individual processes or tasks do not impact other processes or tasks [7]. In order to achieve such Normalized Business Processes (NSBPs), a set of 25 guidelines was developed, based on the four NS theorems, interpreted at the business process level [7].

In subsequent research, NS was extended based on the thermodynamic concept of entropy, initially again focusing on software architectures [8]. As entropy is generally associated with concepts as complexity, amount of disorder or available information, it enables the study of the observability (including detectability and diagnostability) of a (software) system. In statistical thermodynamics, entropy is considered proportional to the number of microstates consistent with one macrostate (i.e., its multiplicity) [13]. The macrostate refers to the whole of externally observable and measurable (macroscopic) properties of a system, corresponding to visible output of a software system (e.g., loggings). The microstate depicts the whole of microscopic properties of the constituent parts of the system, such as binary values representing the correct of erroneous outcome of a task (which we propose to identify based on the concept of "information units", i.e., each unit of processing of which we are interested in independent information about whether it has been executed properly or not). The higher the multiplicity, the more difficult it becomes to identify the precise origin of an observed error. This approach requires a run time view of the system, since macrostates and microstates regard the instantiations of data structures and processing functions [8]. To design information systems exhibiting low entropy, two NS theorems (i.e., separation of concerns and separation of states) have been confirmed, while two additional theorems (i.e., action version transparency and data version transparency) were proposed as well [8].

A similar reasoning based on the entropy definition within statistical thermodynamics, can also be applied to business processes [10], [9], [12]. Again, a business process is considered to be a flow (i.e., including sequences, selections and iterations) of tasks which perform actions on one or more information objects. Considering their execution allows us to define macrostates and microstates on this level as well. The union of values of, for example, the throughput time, quality, resource consumption, quality and executing actors of all task instantiations correspond to a microstate. Given our observability approach, it is proposed to identify a task on the basis of information unit in an organizational context as well. The macrostate of a business process is the (aggregated) information available for an observer (e.g., such as the total throughput or cycle time regarding a process as a whole or any combination of some of its tasks). Multiple microstate configurations consistent with one macrostate (i.e., multiplicity > 1), makes entropy (and the experienced complexity in terms of detectability and diagnostability) increase, and typical management questions more difficult to answer [9]. For instance, it might become unclear which task (or tasks) in a business process was (were) responsible for the extremely slow (fast) completion (of a particular instance) of a business process in case a problematic macrostate is observed (this results in a *detectability* issue). Additionally, as the possibility arises that both problematic and non-problematic microstates result in the same macrostate (e.g., no problem is observed), an *diagnostability* issue might arise as well: while there might be an important and relevant problem in the considered system, it may not catch the attention of the observer. It is clear that both the detectability issue and the diagnostability issue are problematic from a management perspective. Therefore, it becomes logical that organizations can benefit from reducing the amount of entropy present in their business process repository.

No specific guidelines on how to reduce entropy on the level of business processes have been formulated yet. Similar to the software level, it is hypothesized that guidelines to achieve stable business processes will reduce entropy as well. As a first step, we assess in this paper the entropy-reducing capability of the guidelines as formulated by Van Nuffel [7]. More specifically, we investigate whether a violation of each guideline increases the multiplicity (and hence, entropy) of business processes. In case the guidelines for obtaining more evolvable business processes would equally result in business processes exhibiting a lower degree of entropy, the guidelines can obviously be adopted for this latter purpose as well. Also, to the extent that the guidelines can be confirmed from this other theoretical perspective, this provides additional validation of the considered artifact —the set of guidelines— as well. A similar approach (i.e., comparing the guidelines of Van Nuffel [7] with the theoretical framework provided by Enterprise Ontology [14]) was already performed in earlier research ([15], [16]) and proved to be interesting: most of the guidelines of Van Nuffel were found to be consistent or complementary with Enterprise Ontology and only a few of them were considered conflicting.

III. COMPARISON OF GUIDELINES RATIONALES

In this section, we will systematically investigate the guidelines as proposed by the work of Van Nuffel [7]. For each guideline, we will first provide a brief description. Next, we explore whether not adhering to this guideline would imply an increase in entropy as we defined it earlier. Guidelines of which violations result in additional entropy are then considered to be suitable for entropy control as well. To discuss and analyze each of the guideline, we will adopt the same structure as used in ([7], [15], [16]): first a set of general guidelines for identifying business processes will be discussed. Next,

three additional guidelines for specific cases will be analyzed. Finally, some task and auxiliary guidelines are considered.

A. General Business Process Guidelines

The first set of guidelines focuses on the question how to identify a set of tasks as a separate business process.

Guideline 1, "**Elementary Business Process**", requires that a business process should be operating on *one and only one* life cycle information object[1] [7, p. 107]. Not adhering to this guideline would imply a design in which a business process could be operating on multiple life cycle information objects. For instance, consider both invoicing and manufacturing steps which are mixed up and interacting in one process, and a problem with the total throughput time of finishing invoices is present, as represented in Figure 1. At least two situations in which multiplicity > 1 (and entropy arises), can now occur. First, as the business process is concerned with operations on multiple life cycle information objects, the problematic throughput time of the invoicing steps can be "compensated" by "normal" throughput times of the manufacturing steps. Consequently, the problematic total throughput time of the invoicing activities would not necessarily raise an "alert", even after for instance hypothesis testing on the overall observed mean versus expected mean. Therefore, multiplicity > 1 (and entropy increases): the status reflected by the macrostate (e.g., no problems are reported ("OK")), is conform to multiple microstates (e.g., both "OK" or "Not OK" for the throughput time of the invoicing steps). Further, not demanding that business processes operate on a single information object, also implies that multiple business processes can be operating (unconsciously) on identical (not necessarily recognized) information objects (i.e., duplication and copy/paste of (parts of) processes might occur). Therefore, chances that the problematic total throughput time of the invoicing activities would raise an "alert" become even smaller, as the information on this concern is not properly separated or centralized. This situation correlates with our (reduced) detectability interpretation of entropy as pointed out in Section II. Second, in case a problem is observed (i.e., the macrostate signals "Not OK"), multiplicity > 1 as well. The macrostate now complies to multiple microstates: the "Not OK" result of the total throughput time might be related to the manufacturing steps, the invoicing steps or both. In order to diagnose the problem unambiguously, the process owner should disentangle all steps in the business process, determine the life cycle information object they belong to, and analyze to which life cycle information object the overall problem is actually related. Further, we already noted that not demanding a business process to operate on a single information object might result in multiple business processes operating (unconsciously) on identical information objects (i.e., duplication and copy/paste might occur). If the macrostate of multiple business processes (each implementing (duplicate) invoicing steps) goes to "Not OK", chances of identifying "the invoice" as the problematic concern become even smaller, as the information on this issue is not properly separated. This situation correlates with our (reduced) diagnostability interpretation of entropy as pointed out in Section II. Based on these two situations, we can conclude that not adhering to this guideline implies an increased amount of entropy in the business process instantiation space. Therefore, we state that the guideline is suitable for entropy control as well. A small refinement can be formulated by stating that, in order to ensure instance traceability, the specific information object instance a business process instance is operating on, should be stored as an attribute of that business process instance.

Additionally, we note that limiting a business process to a task sequence related to only one life cycle information object also enables entropy reduction in suboptimal cases when the most fine-grained separation of concerns and states at the task level is not performed or deemed feasible. In case the first guideline is adhered but undesirable aggregations at the task level are still performed, an aggregation and entropy depending on the number of combined tasks k occurs. On the other hand, in case multiple life cycle information objects are incorporated into one business process, the amount of entropy becomes dependent on i as well.

Guideline 2, "**Elementary life cycle information object**", defines a *life cycle information object as an information object not exhibiting state transparency* [7, p. 114]. Combined with guideline 1 this implies that a business process is related to one information object not exhibiting state transparency. In this context, an information object is considered state transparent if it adheres to the NS Separation of States principle and the object has no proper state transitions which should be made explicit [7, p. 118]. Not adhering to this guideline would imply two possible situations: (1) the identification of an information object as a life cycle information object when it already exhibits state transparency, or (2) not recognizing a non-state transparent information object as a life cycle information object. Regarding the first situation, the creation of an additional life cycle information object (and a corresponding business process) for an information object of which the states are already fully reflected by another life cycle information object, does neither increase of decrease entropy. No additional information regarding the microstate configuration is retained or lost (the information regarding the states of one particular life cycle information object instance is simply duplicated) by identifying this additional life cycle information object. However, as stated in the discussion regarding the previous guidelines, duplicate process (parts) should be avoided. Regarding the second situation however, an information object not exhibiting state transparency which does not get recognized as a life cycle information object, will generate an increase in the degree of entropy (i.e., multiplicity > 1). As in such case no state transparency regarding the concerning information object is attained, information about its state transitions (and hence, the microstate configuration) is lost. Expressed differently, a multiplicity > 1 will arise during and after execution time as the macroscopic observations regarding this information object cannot be traced to individual tasks represented by states (i.e., a myriad of microstates are possible). This situation relates to both the detectability and diagnostability issues within an entropy viewpoint as pointed out in Section II. Consequently, we can remark this guideline is not strictly necessary to control entropy in the context of

[1] A life cycle information object in this context is to be considered as "an information object whose life cycle is represented by (a) business process(es)" [7, p. 101]. An information object in this context is to be considered as "a concrete, identifiable, self-describing entity of information" that typically has an enterprise-wide unique identity, meaningful to a business user and can contain meta-data that describing its data content [7, p. 100].

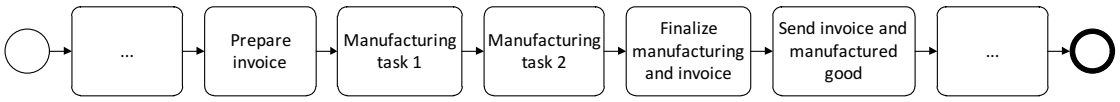


Figure 1. A simple business process operating on multiple life cycle information objects.

the first situation: theoretically speaking, a state transparent information object can be identified as a life cycle information object without increasing entropy (albeit without any thinkable benefit). However, the second situation shows that not adhering to this guideline can imply an increased amount of entropy in the business process instantiation space when a non-transparent information object is not recognized as a life cycle information object. Therefore, we state that the guideline is largely suitable for entropy control and advice its application for this purpose as well. We would further like to add that this guideline actually quite nicely illustrates the core reasoning of designing business processes based on the entropy rationale: for every task of which separate information is valuable (constituting a so-called "information unit"), a separate state should be defined and related to the information object it is operating on. Therefore, each information object not exhibiting state transparency should be considered as a life cycle information object, thereby storing information of each individual task performed on it, at its most fine-grained level.

Guideline 3, "**Aggregated Business Process**", states that in order to represent an aggregated business process, an aggregated life cycle information object has to be introduced (p. 121). This guideline relates to the fact that certain aggregated business processes can be necessary to several reasons. First, the orchestration of different business processes (each operating on a single life cycle information object) by a distinct business process might be necessary. For instance, consider an Order-to-Cash process in which several sub-processes —such as "order entry process", "procurement process", "production process", etcetera— are each individually and successively called, waiting for completion, upon which the next (set of) sub-process(es) is called, completed, etcetera. Second, different (both internal or external) stakeholders might require different perspectives (such as aggregations) due to, for instance, their own functional domain. For instance, in case of very complex business processes, one can imagine that clients or certain actors at a higher management level might be primarily interested in the mere "milestones" (e.g., "order received", "order produced", "order shipped") of a business process, instead of the possible hundreds of more fine-grained states the product might be in during its life cycle. The guideline under consideration prescribes that such *aggregated processes may only be introduced for orchestrating purposes and in case the business processes under consideration are not able to be designed solely based on guidelines 1 and 2*. Once more, not adhering to this guideline would imply two possible situations: (1) designing an aggregated business process while a redesign based on guidelines 1 and 2 would be possible, or (2) not recognizing a business process for orchestrating purposes while a redesign based on guidelines 1 and 2 is not possible. The first situation would clearly imply an unnecessary combination of two concerns and therefore a violation of guidelines 1 and 2 (as a redesign based on them is still possible). Given the fact that both guidelines were proven to mostly result in an increase of entropy when not adhered to, this situation would equally result in an increase of entropy. The second situation would lead to not recognizing a "combined concern": while each of the underlying concerns have their own life cycle information object and corresponding business process, the orchestration or "interfacing" between them can constitute a genuine concern as well. Such orchestration would entail a relevant information unit and is therefore necessary to keep track of when one's aim is to minimize entropy. Imagine an Order-to-Cash process tracking the Order Entry Process, (possibly multiple) Procurement Processes, Production Processes, Delivery Processes, etcetera. While each of these processes clearly designate their own life cycle information object and therefore, business process, the orchestration between them is crucial to be monitored as well. Tracking interfacing issues in this Order-to-Cash Process constitutes relevant information (macroscopically) and in case a customer complains about a lately delivered order (i.e., the macrostate), the specific business process (instance) which is causing this delay (Order Entry, Procurement, etcetera) should be identifiable (i.e., the specific microstate). Not identifying the necessary aggregated process would therefore lead to multiple microstates consistent with one macrostate. This situation relates to both our detectability and diagnostability issues within an entropy viewpoint as pointed out in Section II. We can therefore conclude that not adhering to this guideline implies an increased amount of entropy in the business process instantiation space and state that the guideline is suitable for entropy control as well.

Guideline 4, "**Aggregation Level**", requires that *tasks performed on a different aggregation level should denote a separate business process* (p. 124). An "aggregation level" in this particular guideline is mainly to be understood as focusing on the multiplicities of different information objects (i.e., the different perceived aggregations). For instance, a typical Order within a company might be conceived as being associated with several Product processes, where this Product process at its turn might then again be associated with multiple Part processes. Not adhering to this guideline would imply that it is possible for a business process to execute sequences of tasks situated at different "aggregation levels". Suppose one business process performing a sequence of tasks on a "parent" information object (e.g., "Product") and sequences of tasks on its "child" information objects (e.g., different "Part" instances). As one could argue that such business process is operating on multiple life cycle information objects, our first two arguments are highly parallel to those of guideline 1. First, such business process design would not guarantee that systematic problems regarding, for instance, the overall throughput time of the sequence of tasks performed on the "child" information object are observed. The problematic throughput times might become "compensated" by "normal" throughput times of the other tasks, therefore not necessarily raising an "alert" to the observer. Hence, multiplicity > 1 (and entropy increases): multiple microstates ("throughput times

OK" and "throughput times Not OK") are consistent with one macrostate ("no problems are reported"). This situation correlates with our (reduced) detectability interpretation of entropy as pointed out in Section II. Second, in case a problem is observed (i.e., the macrostate signals "Not OK"), multiplicity > 1 as well. The macrostate now conforms to multiple microstates: the "Not OK" result of the overall process might be related to the sequence of tasks performed on the "parent" information object, the "child" information object or both. This situation correlates with our (reduced) diagnostability interpretation of entropy as pointed out in Section II. Third, no instance traceability regarding the multiple processed Parts within the single business process is feasible in such design. Therefore, the same states regarding the "child" information object sequence are activated several times during the execution of the business process, while in reality dealing with other information object instances. This makes adequate state-tracking (cf. guideline 2) impossible. As a result, the business process owner cannot make the distinction between situations in which the problematic throughput time might be associated with all Part instances in general (i.e., a "systematic" recurring problem) or with one Part instance in particular (and in such case, which specific Product instance). Also in this third situation, this implies multiplicity > 1: one macrostate (i.e., a problem is observed) is consistent with multiple microstate (i.e., the problem is due to Part instance 1, or 2, ..., or all Part instances): certain parts of the microstate configuration are simply not captured during process execution. Based on these two situations, we can conclude that not adhering to this guideline implies an increased amount of entropy in the business process instantiation space. Therefore, we state that the guideline is suitable for entropy control as well.

Guideline 5, "**Value Chain Phase**", states that the *follow-up of an organizational artifact resulting from a value chain phase should denote a different business process* (p. 132). A value chain phase refers to the rather generic, often recurring structure and parts within aggregated business processes in manufacturing organizations (e.g., Order Entry, Procurement, Production, etcetera), such as for instance described by the SCOR reference model. Not adhering to the above described guideline could lead to the following two situations: (1) the steps related to these value chains are incorporated into the aggregated (i.e., orchestrating) business process, or (2) no more grained steps related to each of these value chain phases are discerned and no states regarding them is kept. In the first situation, this would imply a violation of guidelines 1 as multiple life cycle information objects (e.g., Order Entry, Procurement, Procurement) are combined into one business process. Further, guideline 4 would be violated as well because most often, these value chain phases have one-to-many or many-to-many relations. A Customer Order can typically be related to multiple Purchase Orders and/or Production Orders. The second situation would imply violations regarding guidelines 2 (i.e., no life cycle information object is identified for several non-state transparent information objects) and 3 (i.e., an aggregated business process is designed when there are still some opportunities for redesign based on guidelines 1 and 2). A situation in which no relevant states regarding the tasks constituting a value chain phase should be identified, is rather unlikely as this would allow to model almost all necessary activities of a typical manufacturing company within one business process having 5 to 8 tasks. Consequently, as we should earlier how violations regarding guidelines 1 to 4 result in multiple microstates consistent with one macrostate, we can conclude that violating this guideline would generate a multiplicity > 1 as well. Therefore, we state that the guideline is suitable for entropy control as well.

Guideline 6, "**Attribute Update Request**", states that *a task sequence to update an attribute of a particular life cycle information object that is not part of its business process scenarios, is represented by an Attribute Update Request business process* (p. 135). This guideline is subject to two specific conditions. First, it has to concern an update operation for which one single functional task is not sufficient to complete the update request, but rather a sequence (i.e., "process") of activities is required. Second, it concerns update requests which are not part of a branch within the regular business process scenarios. Consequently such procedures can be instantiated several times and during several different "states" of the life cycle of the information object regarding which the update request is actually aimed at. Additionally, such process (verifying for instance the validity of updating a certain information object attribute with a certain new value) will typically differ for each individual attribute. Not adhering to this guideline would imply that tasks for handling an attribute update request, not part of the regular business process scenario, becomes incorporated into the flow of the life cycle information object of which the attribute is requested to be update. Again, such situation can be seen as a violation regarding several of the above mentioned guidelines. Not separating such task sequences would lead to a business process operating on multiple life cycle information objects and —at the same time— one concern being dispersed over several places within one business process (i.e., all the life cycle states in which the update request is allowed), thereby violating guideline 1. Second, the design would make the proper tracking of states impossible as at any point of the business process execution for each time an update request is initiated, the state of the regular business process is suddenly (possibly repeatedly) changed to states regarding this update request (thereby indirectly violating guideline 2). Third, as attribute update requests can be performed several times during one instance of the "parent" business process, both concerns relate in a one-to-many multiplicity, thereby violating guideline 4. Consequently, as we showed earlier how violations regarding guidelines 1, 2 and 4 result in multiple microstates consistent with one macrostate, we can conclude that violating this guideline would generate a multiplicity > 1 as well. Therefore, we state that the guideline is suitable for entropy control as well. From an organizational observability (i.e., entropy) viewpoint, it clearly makes sense to separate such sequence of tasks for future reference. For instance, the calculation of certain measures and the solution for certain managerial questions such as: "how often are such requests accepted/denied and for which reason" or "can we see any relation between the outcome of the update requests and its input values" are only able to be solved in an efficient way when this task sequence is properly separated in its own business process module and not unconsciously repeated in other places throughout the business process repository.

Guideline 7, **Actor Business Process Responsibility**, states that *tasks, of which the task allocation genuinely belongs to a different business process owner, should be designed into a*

separate business process (p. 139). This guideline only applies in very stringent cases. For example, in case legislation or internal audit rules prescribe that different owners should be responsible for other (parts of) task sequences, this guideline applies. Mostly, the guideline is applicable when different parts of a task sequence are performed by different organizations. In such cases, the respective task allocations are logically situated at one of these different organizations as well. From an entropy viewpoint, let us consider the case in which the mentioned guideline is not adhered to. In such case, a business process could consist of a combination tasks which belong to genuinely different business process owners. Each task still has an attribute regarding which actor is allowed or required to perform the task. However, no information is available regarding who is doing the task allocation (e.g., the manager of organization who determines who is doing what). If such information should be retained, the appropriate level is the business process level, as it concerns a sequence of multiple tasks. In case this information is relevant but however no distinct business process would be designed, a multiplicity > 1 (and hence, entropy) arises as one macrostate (e.g., a problem regarding the overall process) complies with multiple microstates (was the task allocation responsibility situated at person A, B, or C?). This situation correlates with our (reduced) diagnostability interpretation of entropy as pointed out in Section II. Therefore, in case the information regarding task allocation responsibility is relevant, a different business process should be identified from an entropy viewpoint to allow for this task allocation responsibility to be traceable. This guideline calls to create an additional level of "process responsibility" (i.e., who allocates tasks among different actors and takes responsibility that they are carried out adequately), in addition to the responsibility for one or multiple tasks. Therefore, we state that the guideline is suitable for entropy control as well. However, in line with the work of Van Nuffel [7] we remark that it should be stressed that identifying additional business processes based on this guideline should be done with extreme precaution to avoid unnecessary additional business processes and, hence, only in cases where a different task allocation responsibility is relevant for observability purposes.

Guidelines 8 and 9 as proposed by Van Nuffel [7], propose two specific business process types to be identified. Guideline 8, "**Notifying Stakeholders**" states that the *communication of a message to stakeholders (in the correct format, incorporating fault handling, etcetera) constitutes a distinct business process* (p. 143). Guideline 9, "**Payment**" states that the *payment of a particular amount of money to a particular beneficiary should equally constitute a distinct business process* (p. 146). Not recognizing these two concerns as distinct business processes could again create two possible situations: (1) integrating the tasks for the notification and payment in other business processes or (2) not specifying their constituting tasks at all. It is clear that the first situation would violate guideline 1 (multiple life cycle information objects operating within one business process) and 4 (for example, multiple notifications can be sent within the scope of one "parent" business process instantiation). The second situation would violate guideline 2 as a non-state transparent information object is not identified as a separate life cycle information object. Consequently, as we showed earlier how violations regarding guidelines 1, 2 and 4 result in multiple microstates consistent with one macrostate, we can conclude that violating this guideline would generate a multiplicity > 1 as well. Therefore, we state that guideline 8 and 9 are suitable for entropy control as well. Designing these task sequences as separate business processes is useful from an organizational observability (i.e., entropy) viewpoint as well. Both the payment of a particular amount in a particular format to a particular beneficiary at the right time, as well as communicating a certain message in a particular format at the right time while maintaining integrity, are often recurring functionalities within typical business processes. As a consequence, due to their frequently occurring nature, a business process owner would typically be interested in certain characteristics of each of these separately recurring tasks sequences: how long do they take to execute, how many times do they result in an error, etcetera. Focusing on these aspects can generate considerable efficiency gains as, for instance, improving the quality metrics or throughput time of the payment process with 5% might entail huge organizational effects as the changes are "expanded" throughout the whole organization. However, these analyses and improvements can only be performed when "payments" and "notifications" are designed into separate business processes. Otherwise, systematic problems regarding one of the concerns might not be noticed (cf. the detectability issue of Section II) or might not be unambiguously traced to the right concern (cf. the diagnostability issue of Section II).

B. Additional Business Process Guidelines

The three additional business process guidelines address decisions on how to identify business process guidelines which are particularly influenced by domain-specific issues.

Guideline 10, "**Product Type**" identifies a Product Type as denoting a separate business process because "*a different type of product / service denotes a main concern*" (p. 149). In this context, a Product Type is considered as an organizational artifact (product, service) sharing a collection of important characterizing properties. Not adhering to this guideline would imply that one business process could combine two or more Product Types. Firstly, this would clearly generate cluttered and poorly organized processes including a large set of branches. However, this could also be considered harmful from our adopted entropy perspective. Imagine a company producing only instances of two Product Types (A and B), but combining all the significantly differing production steps of these two production types into one business process. Suppose that the observed total production costs (i.e., the macrostate) were considered by the manager as too high (i.e., reducing profitability). If in such case the tasks and sequences of tasks to produce both Product Types have been combined and mingled up into one process, it would become difficult to diagnose to which product (A, B or both) the cost problem is related. As a consequence, multiplicity > 1 and entropy increases. This is obviously highly problematic in the context of decisions which need to be taken regarding the product portfolio: which products should the company retain and which products should the company place out of production due to too high costs? Therefore, we state that the guideline is suitable for entropy control as well.

Allocating costs to cost objects such as individual Product Types and their instances is considered key in cost-accounting methods. As an example, the Activity-Based Costing (ABC)

method was designed to more accurately assign indirect costs to individual Product Types using a multi-stage cost allocation procedure involving the identification of activities responsible for the generation of costs [17]. These tasks are identified at multiple levels such as facility, batch, product or unit level. This reasoning is highly related to the identification of business processes at the level of Products, their Parts, etcetera in this paper. However, in [18] we indicated that new ABC iterations does not exhibit the lowest possible degree of entropy. We therefore hypothesize that cost-accounting approaches can benefit from analyzing their proposed methods using an entropy perspective. Additionally, the idea that the modular structure or patterns within organizations should or may reflect the (technical) modular structure of products is not new. For instance, the mirroring hypothesis states that in case of the design of complex systems (such as products), the organizational structure (such as division of labor and division of knowledge) will mirror one another [19]. Considering products as a basis for modularizing cost objects is therefore considered as logical design decision, supporting a low entropy design.

Finally, it should be mentioned that this guideline does not imply that potential Product Types having sequences of tasks in common should repeat these identical sequences in all of their respective business processes. For instance, Product Parts should be modeled in separate business processes and can then potentially be re-used for several other Products (in conformance with guideline 4). Also, variants of one product may be modeled by using gateways and branching options within the process: in such situation the differing tasks should clearly be separated in different tasks and the life cycle information object object data should enable the unambiguous tracing of the variant which was produced (and hence, the business process path which was taken).

Guideline 11, "**Stakeholder Type**" states that a "*stakeholder type should principally be considered a cross-functional concern, except for those business processes where the stakeholder type denotes the life cycle information object, e.g., different HR business processes to deal with different types of employees*". As stated in the guideline, in case each different stakeholder type would be associated with another genuine life cycle information object, another business process type should be identified (in conformance with guideline 1). In such case, the task sequences of the regarding the information object for each stakeholder type significantly differ and distinct information should be kept on this matter. Therefore, not adhering to this part of the guideline would violate guideline 1 and increase entropy. However, in case the stakeholder type merely denotes variants of the product or service to be delivered, the stakeholder type can and should be used as an attribute of the considered life cycle information object. In such case, information is gathered at the level of the information object (e.g., "credit grant") but can be categorized according to the stakeholder type (e.g., "golden type", "normal type") based on this attribute. That way, the identical task sequences are not dispersed throughout the business process repository, thereby avoiding the generation of a higher amount of entropy. In case a particular common sequence of tasks within the business process would be related in a one-to-many or many-to-many relationship to the business processes, this sequence should be separated in its own business process (cf. guideline 4). Likewise, when a particular sequence would be of use within other types of business processes (e.g., a procedure for verifying customer details which is used both in online and desk assisted order entry) or when the stakeholder types are "*completely differently processed*" [7, p. 155], a separate business process should be identified. Indeed, in such cases, separate information regarding these information objects is relevant from an entropy perspective. Therefore, we state that this guideline is generally suitable for entropy control as well. This reasoning is similar to our reasoning regarding guideline 10 on Product Type variants.

Guideline 12, "**Access Channel**" states that the access channels typically denote "*a cross-functional concern*", meaning that no separate business process should be identified (p. 159). In case of small variants within the life cycle it is equally advised from an entropy reasoning to identify one business process regarding, for instance, an Order, in which the branching for the respective access channels can be performed based on an attribute of the life cycle information object. Under the assumption that the differing tasks are clearly separated from the common tasks regarding each considered access channel, this reasoning would be similar as the one mentioned in guideline 10, allowing common task sequences to be properly separated into one process (and therefore not be dispersed throughout the process repository).

However, in some cases, a separate access channel can imply a totally different process in a particular part of a value chain. Different access channels can have different results in terms of throughput time, costs, etcetera. Therefore, it can be interesting to separate business processes based on this concern in certain situations. For instance, consider the application for a business school which can be done online (fill in resume, submit TOEFL, recommendation letters and example paper) or via attendance on site (interview and business game): for both access channels, separate business processes are preferable. If one process would mingle all steps for both access channels (without branching based on an attribute), entropy can occur. Suppose that a supervisor observes that the throughput time of the applications is too lengthy. Now, there is no clear indication whether it is due to the online application channel, the on site application channel, or both. Consequently, multiple microstates are consistent with one macrostate. Therefore, in similar way as cited for guideline 12, we state that the considered guideline is suitable for entropy control in cases when the access channel is used for separating access channel specific tasks among a sequence of common tasks, and for which a selection according to the access channel can be made based on an attribute of the information object (pointing to the access channel used for each instance). The guideline can be further nuanced or refined by stating that, in case totally separate access channels are associated with totally different task sequences (i.e., they are "*completely differently processed*" [7, p. 155]), and of which separate information needs to be available, the creation of a separate business process for each access channel is advised.

C. Task Guidelines

These guidelines focus on the question on how to identify individual tasks within a business process.

Guideline 13, "**A Single Functional Task - Overview**" identifies a task as "*a functional entity of work that either results in a single state transition of a single information object type, or refers to an Update or Read task on a single information object type*" (p. 161). As this guideline provides a general definition of tasks, it should inform us about the conceptualization of "concerns" at the level of individual tasks, regarding which we need to keep track of independent information. This guideline is difficult to be analyzed based on "violations" towards it as it mainly describes how a task is conceptualized, i.e., being a "portion of work" resulting in a single state transition or performing a read/update action. However, we can notice that this conceptualization is rather similar to the definition of a task we adopt in our entropy reasoning (here and in previous publications), although some refinements from the entropy perspective can be made. Remember from Section II that the identification of tasks in an organizational context was equally based on information units, or every part within the life cycle of an information object of which we want to keep independent information. Typically, this information can be expected to be multidimensional in an organizational context: costs, resource consumption, executing actor, consumed time, quality, etcetera. As a consequence, from an entropy point of view, we state that the guideline is suitable for entropy control as well and propose to refine the considered guideline based on these dimensions. This information should be retained in relation to the corresponding state of the considered task. All other guidelines in relation to the identification of tasks are then to be considered as special cases of this guideline.

Guideline 14, "**CRUD Task**" states that "*each of the Create - Read/Retrieve - Update - Delete (CRUD) operations constitutes a single task*" (p. 164). Not adhering to this guideline would imply that CRUD tasks are combined with non-CRUD tasks. Suppose that an error or lacking quality within such flow is observed (i.e., the macrostate) and the observer wants to diagnose the reason for it. In such case, it would not be able to distinguish between situations in which the undesired output of the combined task is due to, for instance, information which was faulty received (i.e., the "Read" action) or due to the action taken based on this information. Consequently, multiple microstates are consistent with one macrostate. From an entropy perspective, it is also interesting to know, for example, who adapted a particular attribute of an information object at which time. Therefore, we state that the guideline is suitable for entropy control as well. In order to ensure instance traceability, we further refine the guideline by stating that the information created, read, updated or deleted should be stored in relation to the state of the considered task instance.

Guideline 15, "**Manual Task**" states that "*every manual task of which the initiation and completion has to be known, has to be designed as a separate task*" (p. 167). Suppose that two or more manual tasks of which the initiation and completion has to be known are combined into one task. This means that only one state regarding this combined task is known. Therefore, the observed macrostate (e.g., the combined task has been completed erroneously) is consistent with a myriad of microstates (e.g., manual task 1, manual task 2 or a combination of both have resulted in this erroneous completion). Therefore, we state that the guideline is suitable for entropy control as well. This guideline should be interpreted in combination with guideline 21, requesting that a task cannot consist of parts that are performed by different actor(s) (roles).

Guideline 16, "**Managing Time Constraint Task**" states that "*the management of a time constraint denotes a separate task because it represents the individual concern of managing a particular time constraint*" (p. 169). Remember that we proposed to refine guideline 13 by requiring that the relevant costs, throughput time, resource consumption, quality criteria and executing actors should be persisted in the state related to each identified task. Since we defined the throughput time of a particular task as a relevant part of information for defining the microstate, this guideline needs to be adhered to from an entropy perspective as well. Consider a particular task for which, after it has been completed, the next task in the sequence is only allowed to proceed in the next morning at 7AM. In case this timing constraint is not properly separated (by a waiting condition or "timer") from the first task, entropy increases as it is not known how long it took to complete the task (and hence, when the actual "waiting" started). In such situation, when for instance trying to reduce the observed occupation time of the production line (i.e., the macrostate), no clear information is available regarding how long the product was actually "occupying" the production line and how long it was waiting for further processing (i.e., multiple microstates). This is clearly associated with an increased amount of entropy. Therefore, we state that the guideline is suitable for entropy control as well.

Guideline 17, "**Business Rule Task**" states that "*a single business rule should be separated as a single task*" (p. 171). In this context, mainly business rules defined as "derivation rules" or "reaction rules" are considered (i.e., deriving a decision —mostly for branching— from other knowledge or based on a particular business event or state). These decisions are clearly something else than the execution of a particular "production task" contributing to the actual product or service (e.g., assembly) realization. Therefore, separate information is required about the executor, time and resource consumption, as well as the final outcome. Suppose the considered guideline is not adhered to and a production activity and business rule activity are combined into one task. In such case, entropy arises due to several reasons. Firstly, unclarity about the typical microstate information dimensions such as time and executor might arise. For instance, "calculating the stock" and "deciding which branch to choose" based on the available stock (e.g., "order" vs. "not order") might be executed by different actor (roles) and therefore have different relevant information. In case this task is executed erroneously or took too long (i.e., the macrostate), it is not clear which actor was responsible or which information unit caused the lengthy throughput time. Clearly, this situation can be avoided by adhering to the some of the other proposed guidelines (such as guidelines 13 and 21). Secondly however, not adhering to this guideline would imply that the eventually chosen branch (i.e., again belonging to the macrostate) is not uniquely traceable to the non-business rule task (e.g., the calculation of the stock) or business rule task (e.g., deciding whether or not to order based on the calculated stock), hence being consistent with multiple microstates (and as a consequence, generating entropy). Therefore, we state that the guideline is suitable for entropy control as well: in some situations, the business rule task will not be separated automatically due to one of the other proposed guidelines, giving the considered guideline its own right to exist.

Guideline 18, "**Bridge Task**" states that "*when a business process instance operating on an instance of life cycle information object type I has to create a business process instance of another life cycle information object type L, this functionality is designed as a bridge task that initiates the creation of the instance of the life cycle information object L, and represents a state transition on the instance of I*" (p. 173). The actor initiating a new instance of a life cycle information object can be different than the actor doing the preceding or following tasks and might send specific configuration data to the instance being created. Therefore, based on guideline 21 (cf. infra), we state that the guideline is suitable for entropy control as well. Potentially, also the time needed to initiate an instance might be time or resource consuming and might therefore require the design of a separate task. Further, based on the need for instance traceability, we propose a refinement of the guideline by stating that the reference to the business process instance operating on the instance of life cycle information object type L should be stored in relation to the state of the bridge task instance. Additionally, the reference to business process instance operation on the instance of life cycle information object type I, as well as the specific configuration data which was used during initiation, should be stored in relation to the instance of life cycle information object L.

Guideline 19, "**Synchronization Task**", states that "*when a business process instance operating on a life cycle information object I has to inform a business process instance of another life cycle information object L, a synchronization task, representing a state transition on the instance of I, alters the state of the business process instance of L*" (p. 176). Whether a Synchronization Task is performed by a Bridge Task to a Notification (see guideline 18) or by an Update Task regarding the state of a particular target life cycle information object (see guideline 14), they both can be considered to be special cases of the respective guidelines. As both of these guidelines have been suggested to be suitable for entropy control, also this guideline should be. Nevertheless, we make two additional remarks on this regard. First, the need for proper state tracking should raise some caution regarding the second option to implement the guideline, i.e., by simply using an update task to alter the state of another life cycle information object. As it should not be allowed to alter the state of a process while another task on the same life cycle information object is been carried out and only valid state transitions are allowed, this implementation should only be chosen with care. Second, to ensure diagnostability, the guideline can be refined by stating that the reference to the business process instance operating on life cycle information object L should be stored in relation to the state of the synchronization task instance. Additionally, the reference to the business process instance operating on life cycle information object I should be stored in relation to the altered state within the business process instance of L.

Guideline 20, "**Synchronizing Task**" calls for identifying separate tasks which receive "*information from another business process's execution, in order to continue the business process control flow*" (p. 178). Whether a Synchronizing Task is performed actively (systematically checking the state of another life cycle information object) or passively (waiting until a Notification is received), this guideline denotes a special case of a waiting condition and therefore of guideline 16. Therefore, we state that also this guideline is suitable for entropy control: non-adherence would imply that the information regarding for example throughput time of other tasks can become misrepresented, thereby generating entropy. A refinement can be formulated by stating that the reference to the business process instance from which the information is received as well as the the incoming information itself, should be stored in relation to the state of the synchronizing task instance.

Guideline 21, "**Actor Task Responsibility**", states that "*a task cannot consist of parts that are performed by different actor(s) (roles)*" (p. 180). Remember that we proposed to refine guideline 13 by requiring that the relevant costs, throughput time, resource consumption, quality criteria and executing actors should be persisted in the state related to each identified task. Therefore, as we defined the actor performing a particular task as a relevant part of information for defining the microstate, it makes sense to support this guideline from an entropy perspective as well. Suppose a task is combining parts A and B which are executed by two different actor(s) (roles). In case a problem regarding the produced product or delivered service is observed afterwards (i.e., the macrostate) and the main problem is traced to the task combining these two parts, it still remains unclear which actor is actually responsible for the lacking quality (the actor performing part A or the actor performing part B), and multiple microstates arise for this single macrostate (thereby increasing entropy). As a result, we state that the guideline is suitable for entropy control as well and can even be refined and formulated more strictly as "A task cannot consist of parts that are performed by different actor(s) (roles) and the specific actor performing the task should be persisted in its associated state" in order to ensure instance traceability.

D. Auxiliary Guidelines

Auxiliary guidelines do not specifically focus on identifying tasks or business processes, but try to formulate a set of generally applicable guidelines to design business process repositories.

Guideline 22, "Unique State Labeling", states that "*each state of a life cycle information object has to be unique*" (p. 181). This guideline follows directly from the NS Separation of Concerns and Separation of States principles. In case this guideline would not be adhered to, multiple states could be attributed the same identifier and would obviously defy the benefit of introducing states in order to reduce the amount of entropy. Consider for instance a process in which the results of two distinct consecutive manufacturing steps (on which separate information is relevant) are persisted in the same state. In case for instance the quality of the resulting product is insufficient (i.e., the observed macrostate), it is uncertain which of the considered manufacturing steps is responsible for this failure (i.e., meaning that at least two microstates are consistent with the observed macrostate, thereby generating entropy). Therefore, this guideline is consistent with efforts to reduce the entropy generated by executed business processes. We refer again to the refinement of guideline 13 which indicates the different information dimensions which should be related to such states.

Guideline 23, "Unique State Property", states that "*a life cycle information instance can only be in a single state*

at any time" (p. 182). Also this guideline follows directly from the NS Separation of Concerns and Separation of States principles. When business processes are conceived similarly as "production lines" operating on life cycle information objects, each instantiation should obviously be in one state at the time. In case this guideline would not be adhered to, this would mean that a business process can be in two states at one point in time. For instance, this would allow an instance of a payment life cycle information object to be in the state "initiated" and "finished" at the same time. This would make any traceability of observed macrostates in terms of microstates impossible as, for example, no clear information can be retrieved on how long it took to complete one particular invoice (i.e., going from the state "initiated" to the state "finished"). Clearly, abundant entropy would arise and the guideline should be adhered to in order to avoid this.

Guideline 24, "Explicit Business Process End Points", states that "*if a business process type has multiple possible outcomes, each of these scenarios should have its dedicated end point reflecting the respective end state of a business process instance*" (p. 183). This guideline equally follows directly from the NS Separation of Concerns principle. In fact, it could be considered as a special case of guideline 22. Consider two different outcomes (e.g., due to two different microstates by different branches) resulting in the same state (i.e., the observed macrostate), e.g., "process finished". In such situation, two microstates would —by definition— be consistent with one macrostate and entropy occurs. Also, it would become impossible to analyze how many times, in case of claim handling for instance, the process results in "claim successfully handled" and "claim not successfully handled", or how these states have come into being. Therefore, also this guideline is consistent with the aim of reducing entropy generation.

Guideline 25, "Single Routing Logic", states that "*a split/join element in a business process's control flow should only represent a single split or join routing expression*" (p. 184). Not adhering to this guideline would imply that multiple elements could be combined into one module, i.e., both joint and split conditions. However, given the convention that states can only be related to tasks (not gateways), not adhering to this guideline could generate entropy. Combining split/joint elements into one module would prohibit the creation of an intermediate task between the join and split, thereby assuming that the logic or business rule for deciding which branch to take in the split element is incorporated in the gateway (for which it is not intended as it will also not result in a persisted state). Therefore, no information regarding this branching decision (e.g., evaluate number of parts in stock) is persisted in a state. In such situations one macrostate (the observed outcome and chosen branch) is consistent with multiple microstates (it is unclear who has taken this decision, based on which information, requiring how many resources, needing which amount of time, etcetera), thereby generating entropy. Therefore we state that also this guideline is consistent with the aim of reducing entropy generation during the execution of business processes.

IV. Discussion, limitations and future research

This paper aims to contribute to our research line on how to prescriptively design business processes regarding certain criteria (such as low complexity and high evolvability). In earlier work, a set of prescriptive guidelines has been proposed from the stability perspective [7], and the applicability of the entropy concept to study the observability of business processes has been reported [9], [10]. The main focus in this paper was to verify whether the already existing guidelines to optimize the business process design from a stability viewpoint align with the perspective to minimize entropy. We found that most of the investigated guidelines are rather consistent among both approaches: guidelines required to attain evolvability enable observability and vice versa. This is illustrated in Table I. Regarding the general business process guidelines, some small exceptions were noticed for guidelines 1, 2 and 7. For guideline 1, it was stated that instance traceability required that the specific information object instance a business process instance is operating on, should be stored as an attribute of the latter. For guideline 2, it was observed that —theoretically— entropy does not increase when a state transparent information object is identified as a life cycle information object. For guideline 7, it was argued that the application of the specific guideline should be performed even more thoughtfully and exceptionally when one is adopting the entropy viewpoint as its necessity in many situation is not really compelling. Regarding the additional business process guidelines, we made an additional nuance regarding the possible identification of a separate business process based on its access channel (cf. guideline 12). While we in general advise to follow the guideline, we added some additional remark and clarification on possible cases in which different access channels could still depict another separate business process. Regarding the task guidelines, we concluded that the guidelines of Van Nuffel were consistent with our proposed entropy reasoning, although we were able to propose some additional refinements of the guidelines based on this new perspective. For guideline 13, we stressed the multidimensional nature of the information to be stored in relation to a state. For guidelines 14, 18, 19, 20 and 21 we proposed some small refinements each related to the need for instance traceability. Finally, all auxiliary guidelines were deemed consistent with our entropy reasoning.

The conclusion that the guidelines from the stability point of view correlate with these from the entropy viewpoint is encouraging for business process designers and researchers, as this might indicate that a unified set of business process design guidelines might be conceivable, optimizing multiple important design characteristics concurrently. Nevertheless, to a certain extent, this conclusion might come as a surprise as well, given the different assumptions and analysis viewpoints of both approaches. First, while both approaches do not only take a different perspective towards business process analysis (i.e., search for evolvability vs. observability), they take a fundamentally different perspective for obtaining their goal as well. The evolvability analysis focuses on the mere *design time* of business processes, which means that the harmful effects its aims to resolve (the so-called "combinatorial effects") are situated on this perspective: a functional change that causes N changes in the business process design. In contrast, the observability analysis focuses on avoiding harmful effects during *execution time*: a multiplicity > 1 (which we could coin as an "uncertainty effect") only manifests itself when the business processes are executed. Clearly, these effects are caused by choices made at design time. However, as

TABLE I. AN OVERVIEW OF THE ANALYSIS OF THE GUIDELINES OF VAN NUFFEL [7] FROM THE ENTROPY VIEWPOINT

Guideline		Contradicting	Compliant	Refined
1	Elementary Business Process			•
2	Elementary Life Cycle Information Object			•
3	Aggregated Business Process		•	
4	Aggregation Level		•	
5	Value Chain Phase		•	
6	Attribute Update Request		•	
7	Actor Business Process Responsibility			•
8	Notifying Stakeholders		•	
9	Payment		•	
10	Product Type		•	
11	Stakeholder Type		•	
12	Access Channel			•
13	A Single Functional Task - Overview			•
14	CRUD Task			•
15	Manual Task		•	
16	Managing Time Constraint		•	
17	Business Rule Task		•	
18	Bridge Task			•
19	Synchronization Task			•
20	Synchronizing Task			•
21	Actor Task Responsibility			•
22	Unique State Labeling		•	
23	Unique State Property		•	
24	Explicit Business Process End Point		•	
25	Single Routing Logic		•	

the harmful effects are only visible at execution time, this perspective has to taken into account for this particular analysis as well. It is therefore important to note that, to the best of our knowledge, few modeling languages in the business process modeling domain are currently available to pursue this goal. While many business process modeling notations (e.g., BPMN) allow for a design time analysis of business processes, execution time analysis and visualization of executed business processes in terms of the data they generate is under explored and few starting points are available. We therefore encourage the business process research community to elaborate further on this issue.

Second, the criteria both approaches use to delineate and identify the different business processes and their constituting tasks, differ. The evolvability approach employs the concept of "*change drivers*" (i.e., parts within the business process design which are assumed to change independently) to identify and isolate concerns, whereas the complexity approach employs the concept of "*information units*" (i.e., these parts within the business process design of which independently traceable information is assumed to be needed later on). In our view, it makes sense to state that change drivers are information units and vice versa. In many organizational fields, it it generally accepted that those things you want to change ("change drivers"), have to be measured first (i.e., in order to detect whether a problem in fact exists, and to obtain a reference point to evaluate afterwards whether the changes resulted in improvements) and should hence be recognized as "information units". On the other hand, one could argue that measuring parts of business processes on which you track independently traceable information (i.e., "information units") only makes sense if you are able to potentially change (hence, ameliorate) them later on (i.e., they need to be "change drivers" as well). It is therefore hypothesized that in general, the concerns which should be used to delineate and identify business processes or tasks are determined by the union of "change drivers" and "information units". This hypothesis also allows that, while most of the concerns are expected to be mutual for both perspectives, some of them can actually be derived from only one perspective, but not contradicting the reasoning of the other one. This was for example the case for guideline 2 which was proposed from the stability point of view but is only necessary in a refined way from the entropy point of view. Given the additional, more in-depth analysis of the entropy approach by incorporating the execution time perspective (e.g., the importance of observability), additional concerns which do not seem necessary from the evolvability perspective, might be potentially identified in future research. For instance, the entropy point of reasoning at the software level of NS, proved to suggest additional principles [8].

Notwithstanding the limitations and need for future research, this paper can claim a number of contributions. First, we further contributed to the enterprise and business process engineering field by elaborating on the usefulness to take an entropy perspective for studying the complexity of business processes. Second, we validated the suitability of a set of (already existing) business process design guidelines in this context as a first step towards a Design Theory [20]. In literature, it is generally acknowledged and even encouraged that such design efforts are guided by principles from related scientific fields (i.e., "kernel theories") [21], such as the concept of entropy from thermodynamics. Third, we adopted logical reasoning as our evaluation method, which is one of the suitable validation methods proposed within Design Science research and adopted by several authors [22]. While one can argue that this validation method is not necessarily the most powerful one, we believe that many other validation methods are less suitable for the artefact under consideration. For instance, comparing the results of applying entropy reducing business process design guidelines with other approaches is difficult as few or no prescriptive guidelines for business processes are available, certainly if one is looking for entropy reducing methods.

In future research, we could focus on attempts to verify whether guidelines can be found are necessary from the entropy viewpoint, but are not required from the stability point of view (e.g., related to detectability and diagnostability issues at execution time). Further, although an initial case study (i.e., simulation) has already been performed earlier to show the relevance and applicability of the entropy viewpoint to analyze business processes [12] and six case studies have been documented regarding the design of business processes from the evolvability viewpoint [7], additional cases might be beneficial to specifically illustrate the differences between both approaches and the grounding for their respective business process design guidelines. Additionally, this way of working can further complement our currently adopted evaluation method of logical reasoning. Finally, similar to NS reasoning [6], the ultimate goal of the research stream is to look for organizational "elements" [23], [24]: groupings or patterns of frequently occurring, rather general business processes exhibiting both a high degree of evolvability and observability (i.e., high detectability and diagnostability of business process problems and outcomes).

V. Conclusion

Contemporary organizations need to be agile regarding both their IT systems and organizational structures (such as business processes). Normalized Systems theory has recently proposed an approach to build evolvable IT systems, based on the systems theoretic concept of stability. However, its applicability to the organizational level, including business processes, has proven to be relevant in the past and resulted among others in a set of 25 guidelines for designing business processes. This paper investigated the validity of these guidelines from another theoretical perspective, more specifically, entropy as defined in statistical thermodynamics. We concluded that most of the investigated guidelines are consistent among both approaches: guidelines required to attain evolvability also enable observability (i.e., low entropy) and vice versa. Part of one guideline was found to be not strictly necessary from an entropy viewpoint (only from the evolvability viewpoint), but not contradicting entropy minimization either. Several guidelines were able to be refined to some extent based on our entropy reasoning or were subject to some additional nuancing. Future research should be directed towards entropy specific guidelines, additional case studies and patterns at the organizational level.

Acknowledgment

P.D.B. is supported by a Research Grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

References

[1] P. De Bruyn, D. Van Nuffel, P. Huysmans, and H. Mannaert, "Confirming design guidelines for evolvable business processes based on the concept of entropy," in Proceedings of the Eighth International Conference on Software Engineering Advances (ICSEA), 2013, pp. 420–425.

[2] E. Overby, A. Bharadwaj, and V. Sambamurthy, "Enterprise agility and the enabling role of information technology," European Journal of Information Systems, vol. 15, no. 2, 2006, pp. 120–131. [Online]. Available: http://dx.doi.org/10.1057/palgrave.ejis.3000600

[3] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven process modeling guidelines (7pmg)," Inf. Softw. Technol., vol. 52, no. 2, Feb. 2010, pp. 127–136. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2009.08.004

[4] L. Brehm, A. Heinzl, and M. Markus, "Tailoring erp systems: a spectrum of choices and their implications," in Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001.

[5] H. Mannaert, J. Verelst, and K. Ven, "The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability," Science of Computer Programming, vol. 76, no. 12, 2011, pp. 1210–1222, pdf.

[6] ——, "Towards evolvable software architectures based on systems theoretic stability," Software: Practice and Experience, vol. 42, no. 1, January 2012, pp. 89–116, pdf.

[7] D. Van Nuffel, "Towards designing modular and evolvable business processes," Ph.D. dissertation, University of Antwerp, 2011.

[8] H. Mannaert, P. De Bruyn, and J. Verelst, "Exploring entropy in software systems : towards a precise definition and design rules," in The Seventh International Conference of Software Engineering Advances (ICSEA), 2012, pp. 84–89.

[9] P. De Bruyn, P. Huysmans, G. Oorts, and H. Mannaert, "On the applicability of the notion of entropy for business process analysis," in Proceedings of the Second International Symposium on Business Modeling and Software Design (BMSD), 2012, pp. 128–137.

[10] P. De Bruyn, P. Huysmans, H. Mannaert, and J. Verelst, "Understanding entropy generation during the execution of business process instantiations: An illustration from cost accounting," in Advances in Enterprise Engineering VII, ser. Lecture Notes in Business Information Processing, H. Proper, D. Aveiro, and K. Gaaloul, Eds. Springer Berlin Heidelberg, 2013, vol. 146, pp. 103–117.

[11] P. De Bruyn and H. Mannaert, "On the generalization of normalized systems concepts to the analysis and design of modules in systems and enterprise engineering," International journal on advances in systems and measurements, vol. 5, 2012, p. 3/4.

[12] P. De Bruyn, P. Huysmans, and H. Mannaert, "A case study on entropy generation during business process execution: a monte carlo simulation of the custom bikes case," in Proceedings of the Third International Symposium on Business Modeling and Software Design (BMSD), 2013.

[13] L. Boltzmann, Lectures on gas theory. Dover Publications, 1995.

[14] J. Dietz, Enterprise Ontology: Theory and Methodology. Springer-Verlag Berlin Heidelberg, 2006.

[15] P. Huysmans, D. Van Nuffel, and P. De Bruyn, "Consistency, complementalness, or conflictation of enterprise ontology and normalized systems business process guidelines," in Proceedings of the Third International Symposium on Business Modeling and Software Design (BMSD), 2013.

[16] D. Van Nuffel, P. Huysmans, and P. De Bruyn, "Engineering business processes: comparing prescriptive guidelines from eo and nsbp," Lecture Notes in Business Information Processing (LNBIP), 2014, in press.

[17] C. Drury, Management and Cost Accounting. Sout-Western, 2007.

[18] P. Huysmans and P. De Bruyn, "Activity-based coscost as a design science artifact," in Proceedings of the 47th Hawaii International Conference of Systems Sciences (HICSS), 2014, pp. 3667–3676.

[19] L. Colfer and C. Baldwin, "The mirroring hypothesis: Theory, evidence and exceptions," Harvard Business School Working Paper, 2010.

[20] S. Gregor and D. Jones, "The anatomy of a design theory," Journal of the Association for Information Systems, vol. 8, no. 5, 2007, pp. 312–335.

[21] J. Walls, G. Widmeyer, and O. El Saway, "Building an information system design theory for vigilant eis," Information Systems Research, vol. 3, no. 1, 1992, pp. 36–59.

[22] V. Vaishnavi and W. Keuchler, Design Science Research Methods and Patterns: Innovating Information and Communication Technology. Auerbach Publications, 2008.

[23] P. De Bruyn, "Towards designing enterprises for evolvability based on fundamental engineering concepts," in On the Move to Meaningful Internet Systems: OTM 2011 Workshops, ser. Lecture Notes in Computer Science, R. Meersman, T. Dillon, and P. Herrero, Eds. Springer Berlin Heidelberg, 2011, vol. 7046, pp. 11–20.

[24] P. De Bruyn, H. Mannaert, and J. Verelst, "Towards organizational modules and patterns based on normalized systems theory," in Proceedings of the Ninth International Conference on Systems (ICONS), 2014, pp. 106–115.

A Framework for Autonomic Software Deployment of Multiscale Systems

Raja BOUJBEL
Université de Toulouse
UPS - IRIT
118 Route de Narbonne
F-31062 Toulouse, France
Raja.Boujbel@irit.fr

Sébastien LERICHE
Université de Toulouse
ENAC
7 Avenue Édouard Belin
F-31055 Toulouse, France
Sebastien.Leriche@enac.fr

Jean-Paul ARCANGELI
Université de Toulouse
UPS - IRIT
118 Route de Narbonne
F-31062 Toulouse, France
Jean-Paul.Arcangeli@irit.fr

Abstract—**Automated deployment of software systems in pervasive and open environments is an open issue. There, the topology of target hosts is not always known at design time due either to unforeseen hardware limitations or failures (network links, hosts, etc.) or to device arrival and disappearance. Rather than manually building and executing a static deployment plan, as it is usually done, our approach promotes the specification of deployment properties (requirements and constraints), then their handling by a middleware for autonomic deployment. This paper presents MuScADeL, a new domain-specific language designed to support the expression of properties related to multiscale and autonomic software deployment. It also presents a chain of software tools that participate in the deployment process and are part of the autonomic deployment middleware, including a system of probes for the monitoring of the host machines and a compiler of multiscale deployment properties.**

Keywords—**Software deployment, multiscale distributed systems, domain-specific language, autonomic computing, constraint satisfaction problem.**

I. Introduction

Pervasive computing, on the one hand, and cloud computing, on the other hand, are central topics in several recent research studies. Contributions in both domains have reached a good level of maturity. Nowadays, new research works have identified the need to make pervasive and cloud computing systems collaborate, so as to build systems which are distributed over several scales, called "multiscale" systems. In multiscale systems, decentralization, autonomy and adaptiveness are essential features.

In this context, our work focuses on software deployment and our goal is to develop a framework for supporting the deployment of multiscale applications. Deployment aims at making and keeping software systems available for use, in a situation of mobility, openness and variability of the quality of the resources. Deployment strategies should take into account the multiscale aspects like geography, network, device, and user, as well as non functional properties such as efficiency and privacy.

In this paper, we describe a Domain-Specific Language (DSL) dedicated to multiscale and autonomic software deployment, named MuScADeL (***Mu**lti**Sc**ale **A**utonomic **De**ployment **L**anguage*) [1], then we present how the deployment plan can be computed from the MuScADeL specification.

In the rest of this section, the novel concept of multiscale system and the basics of software deployment are introduced, then the problem of multiscale software deployment is analyzed, and the requirement of a DSL is expressed. Finally, Section I-E presents the plan of the article.

A. Multiscale distributed systems

The term "multiscale system" is present in several recent research papers [2], [3], [4]: in these works, authors consider to make collaborate very small systems (objects from the Internet of Things paradigm as, for example, swarms of tiny sensors with very low computing capabilities) with very big systems (such as those found in cloud computing). They agree that new issues arise, mainly those related to huge heterogeneity.

The INCOME project [5] aims at designing software solutions for context management in multiscale systems, that is to say not only in ambient networks, but also in the Internet of Things and the Cloud, able to operate at different scales and to deal with the passage from a scale to another one. Context management is a complex service in charge of the gathering, the management (processing and filtering), and the presentation of context data to applications, which realization is distributed on the different devices which compose the system. So, context managers are open multiscale applications, and we are interested in their deployment.

In [6], Rottenberg *et al.* argue that the multiscale nature of a distributed system should be analyzed independently in several specific viewpoints such as geography, network, device, data, user, etc. Thus, a distributed system can be described as multiscale when, given a viewpoint, for at least one dimension of this viewpoint, the elements of its projection onto this dimension are associated with different scales. Fig. 1, extracted from [7], shows an example of scales in the "Processing power" dimension in the "Device" viewpoint. The dimension "Processing power" is composed by several scales: kilo scale, giga scale, and peta scale. A family of device can be contained in one scale, as personal devices in the kilo

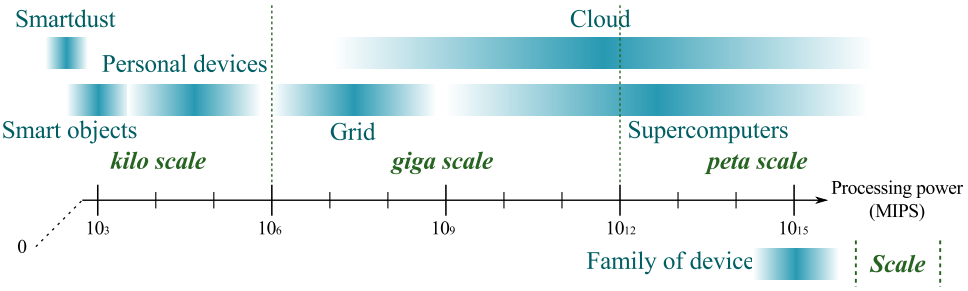


Fig. 1: Scales in the "Device Processing power" dimension.

scale, or in more than one scale, as supercomputers in giga and peta scales.

However, the concept of "multiscale system" is not actually mature. The construction of future multiscale distributed systems will necessitate new kinds of languages, middleware and patterns, allowing to take in consideration the multiscale aspects of the systems.

B. Software deployment

Software deployment is a post-production process which consists in making software available for use and then keeping it operational. It is a complex process that includes a number of inter-related activities such as installation of the software into its environment (transfer and configuration), activation, update, reconfiguration, deactivation and deinstallation [8]. Fig. 2 represents the sequence of the activities. Software release and software retire are carried out on the "producer site", while the other activities are carried out on the "deployment site", some of them at application runtime.

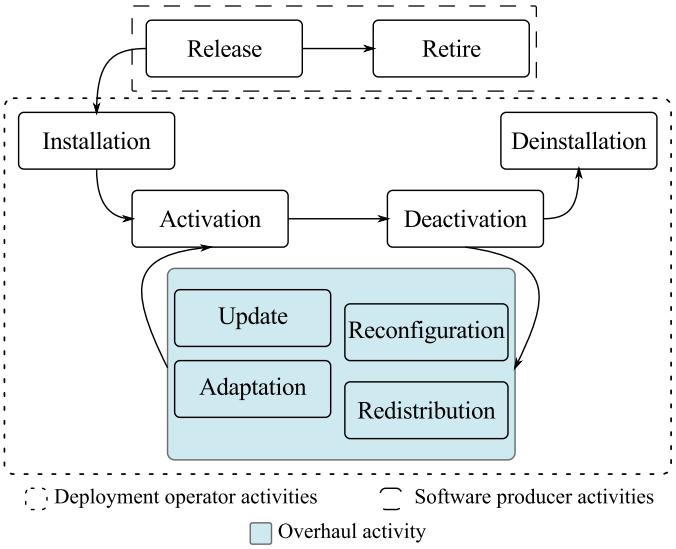


Fig. 2: Software deployment life cycle.

Deployment design is handled by an engineer called "deployment designer". He has to gather information not only about the software system to deploy and the properties of each of its components but also about the distributed organization of the software at runtime. Designing deployment may consist in expressing requirements and constraints. For instance, the deployment designer may express that a particular software component should be installed on some specific devices or on any device, even on incoming ones in case of dynamic systems, while satisfying a set of properties. As a concrete example, consider a software component C which should be deployed on each smartphone which runs Android, has the GPS function active, and is connected by WiFi.

A deployment plan is a mapping between a software system and the deployment domain, increased by data for configuration (and about dependencies). The deployment domain is the set of networked machines or devices which hosts the components of the deployed software system. The ultimate purpose of deployment design is to produce a deployment plan which complies with the expressed properties. Usually, this task is undertaken by a human actor.

At runtime, software must be deployed on the domain according to the deployment plan, this task being possibly undertaken or controlled by an operator called "deployment operator". Automatization of deployment aims at avoiding (or limiting) human handling in the management of deployment.

Fig. 3 shows the timeline of deployment.

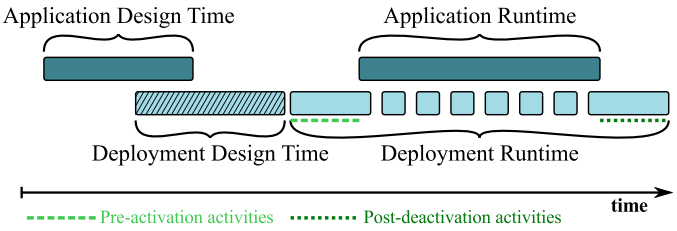


Fig. 3: Software deployment timeline.

C. Multiscale software deployment

In this work, we focus on deployment design, and particularly on the ways for a deployment designer to express multiscale deployment properties.

Software deployment in large-scale and open distributed systems (such as ubiquitous, mobile or peer-to-

peer systems) is still an open issue [9]. There, existing tools for software deployment are reaching their limits: they use techniques that do not suit the complexity of the issues encountered in such infrastructures. Indeed, they are only valid within fixed network topology and do not take into account neither host and network variations of quality of service nor failures of machines or links which are typical of these environments.

Moreover, users of the deployment tools are required to manage manually the deployment activities, which needs a significant human involvement, possibly out of reach of concerned end-users (for example, in case of personal devices like smartphones): for large distributed component-based applications with many constraints and requirements, it is too hard and complicated to accomplish the deployment process manually. Consequently, there is a need for new infrastructures and techniques that automate the deployment process and allow a dynamic reconfiguration of software systems with few or without human intervention.

Additionally, in our opinion, decentralization, openness and dynamics (mobility, variations of resources availability and quality, disconnections, failures) are in favor of autonomy: the autonomic computing approach [10], where the system self-manages some properties (self-configuration, self-healing), may support solutions which satisfy the requirements of distributed multiscale software systems deployment. This idea lead us to "autonomic software deployment" [9].

Instead of directly expressing a statically defined deployment plan, we propose to express *properties*: deployment *requirements* and component *constraints* from which the deployment plan can be computed. In this paper, we focus on the expression of the properties, and on the construction of the plan.

So, in order to build the plan, and moreover to allow management of deployment at runtime, data about the domain must be collected. Thus, a system of probes should run and collect data ranging from the domain properties such as free RAM to more abstract ones related to multiscale (viewpoints, dimensions, and scales). Relations between probes and properties can be made explicit at the same level as the deployment properties in order to allow the specification of the system of probes at the deployment design time.

D. Towards a DSL for autonomic software deployment of multiscale systems

In this ongoing work, our aim is to provide a solution for the expression of the deployment design, concerning in particular the scales and other significant properties of multiscale software systems (see below an example in Section III).

Deployment is a specific operation on software. Its design requires particular skills. Thus, we think that the deployment designer could benefit from a dedicated language when stating the properties. So, we propose a DSL dedicated to the description of deployment constraints and requirements. DSLs present several advantages: they use idioms and abstractions of the targeted domain, so they can be used by domain experts; they are light, so easy to maintain, portable, and reusable; they are most often well documented, coherent and reliable, and optimized for the targeted domain [11], [12], [13].

E. Plan of the article

The rest of the paper is structured as follows. Section II discusses related work on DSL-based software deployment. Section III provides an example of deployment of a multiscale software system. The DSL MuScADeL is presented in Section IV using the example presented in Section III. Section V introduces the software elements that complement MuScADeL in order to compute a deployment plan. Section VI presents the bootstrap of the deployment management system, its architecture, and the interface used by the deployment operator. Section VII explains how deployment properties are transformed and formalized. Section VIII presents our constraint solving library and its use through the MuScADeL specification presented in Section IV. Section IX concludes and discusses some future works.

II. Related work on DSL-based software deployment

The need for automation in software deployment has given to this activity a special attention both in academia and in industry. There are a large number of tools, procedures, techniques, and papers addressing different aspects of the software deployment process from different perspectives.

Existing deployment platforms propose several formalisms to express deployment constraints, software dependencies, and hardware preferences of software to deploy. Usually, the formalisms include architecture description languages (ADL), deployment descriptors (like XML descriptor deployment), and dedicated languages (DSL). In this section, we overview some works on software deployment that propose the use of a DSL.

Fractal Deployment Framework (FDF) [14] is a component based software framework to facilitate the deployment of distributed applications on networked systems. FDF is composed of a high-level deployment description language, a library of deployment components, and a set of end-user tools. The high level FDF deployment description language allows end-users to describe their deployment configurations (the list of software to deploy and the target hosts). Finally, FDF provides a graphical user interface allowing end-users to load their deployment configurations, execute and manage them. The deployment unit is an archive that contains the software binary and the deployment descriptor. The

main limitation of this tool is the static and manual attributes of the deployment. Although the static deployment plan is eligible in a stable environment like Grid, this deployment is not usable in an environment characterized by a dynamic network topology such as ubiquitous environments. Another limitation is that in runtime this tool does not provide mechanisms for dynamic reconfiguration which allows the treatment of the hosts and the network failures.

Dearle *et al.* [15], [16] present a framework for autonomic management of deployment and configuration of distributed applications. To facilitate the work of the deployment designer, they define a DSL, Deladas. Using it, a set of available resources and a set constraints are specified. These definitions permit to generate an applicable deployment plan. The constraint-based approach avoids the deployment designer specifying precisely the location of each component, and then rewriting all the plan in case of problems with a resource. Deladas does not allow to express multiscale properties and constraints. Openness is neither taken into account, the set of hosts is statically defined in a file by the deployment designer. Deployment is still autonomic: at runtime, when the deployment middleware detects a constraint violation (dependencies between components), it tries to solve it by a local adaptation. The new deployment plan is computed by a centralized management component called MADME.

Matougui *et al.* [9] present a middleware framework designed to reduce the human cost for setting up software deployment and to deal with failure-prone and change-prone environments. This is achieved by the use of a high-level constraint-based language and an autonomic agent-based system for establishing and maintaining software deployment. In the DSL called j-ASD, some expressions dedicated to deal with autonomic issues are proposed. But they target large-scale or dynamic environments such as grids or P2P systems, only within the same network scale.

Sledviewsky *et al.* [17] present an approach that incorporates DSL for software development and deployment on the cloud. Firstly, the developer defines a DSL in order to describe a model of the application with it. Secondly, the application is described using the DSL, then it is translated into specific code and automatically deployed on the Cloud. This approach is specific to the deployment of a Web application on the cloud. It highlights the need to facilitate the work of the deployment designer, and that using DSL is a solution for that.

Recent works around software deployment start taking into account constraints of quality of service. For example, Malek *et al.* [18] present a framework (tools and formalism) aiming at determining a ”best” deployment plan regarding several constraints of quality of service which can be contradictory.

Thus, existing solutions for DSL-based autonomic software deployment does not allow deployment designers to express properties related to multiscale concerns, or only in a limited way concerning some scales from the network or device viewpoints. Additionally, dynamics and openness are not or little considered. Even if MADME deals with the dynamics of the deployment domain, the adaptation of the deployment plan is centralized. Finally, the solutions do not define a complete workflow from the design to the fulfilment of the deployment plan while taking into account the current operational context.

III. Example of the deployment of a multiscale software system

In this section, we present an example of the deployment of a multiscale software system, in order to illustrate our aim. Let’s consider a software system made of different components, each of them having specific individual runtime constraints (memory, OS, etc.). The deployment designer may want to express not only these constraints, but also some requirements related to the distribution of the components. For instance, the deployment designer may want that (C1…C6 are software components):

- a resource-consuming component C1 runs on a cloud,
- C2 runs on several machines in a given geographical area, *e.g.*, a city,
- C3 runs on the same type of device than C1,
- C4 runs on any smartphone of the domain,
- C5 runs on the same network than C4,
- C5 number of deployed instance is relative to C4 instances, *i.e.*, for three instance of C4 on intance of C5 is deployed,
- C4 runs on any new smartphone entering in the domain at runtime,
- C6 runs on one machine on each city.

Moreover, some components may have constraints to run properly, such as:

- C1 requires that the component C0 is installed and activated locally,
- C2 must run on a Linux OS and an Arduino (single-board micro-controller) must be connected to the hosting device,
- C3 requires 40M of free RAM at activation time (Freespace),
- C5 requires a 100G hard drive (HDSize).

Fig. 4 illustrates such an example.

IV. MuScADeL: a DSL for Multiscale Autonomic Deployment

In this section, we describe by means of an example MuScADeL, our proposition of a DSL dedicated to the autonomic deployment of multiscale distributed systems. Tokens and keywords are presented further

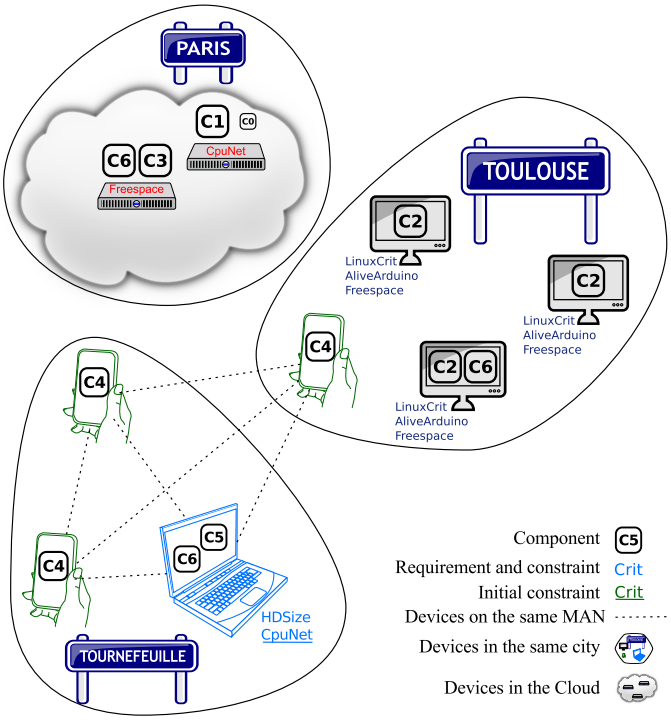


Fig. 4: Example of multiscale deployment.

and the grammar is defined in EBNF syntax (cf. Appendix A). The last version of the grammar is available at http://anr-income.fr/T5/ebnf-muscadel.html.

A. Elements of the language

We present and explain the main elements of MuScADeL language using as example the code for the deployment of the multiscale distributed software system presented in Section III.

1) Component: The keyword **Component** defines a component (cf. Listing 1). The **Version** field is useful for the update activity. The **URL** field specifies the address where the component is reachable for download. The **DeploymentInterface** field specifies the interface of the component, necessary for the interactions with the deployment system: the latter must interact with the component, for configuring and starting it, for managing it at runtime, and for stopping it. The **Dependency** field lists required components: when installing the component, the deployment system checks that whether the required components are installed, or if not, installs them. The **Constraint** field lists hardware and software criteria (defined using the keyword **BCriterion**, see Listing 3) that the component must satisfy. By default, these constraints are permanent —*i.e.*, they must be satisfied both when generating the deployment plan and at runtime — so, the deployment system must check that there is no constraint violation at runtime. For the keyword **InitOnly**, see *6)*.

```
1  Component C0 {
2    Version 1
3    URL "http://test.fr/plopC0.jar"
4  }

6  Component C1 {
7    Version 1
8    URL "http://test.fr/plopC1.jar"
9    Dependency C0
10   DeploymentInterface fr.enac.plop.DIimpl
11 }

13 Component C2 {
14   Version 1
15   URL "http://test.fr/plopC2.jar"
16   DeploymentInterface fr.enac.plop.DIimpl
17   Constraint Freespace LinuxCrit ActiveArduino
18 }

20 Component C3 {
21   Version 1
22   URL "http://test.fr/plopC3.jar"
23   DeploymentInterface fr.enac.plop.DIimpl
24   InitOnly Constraint Freespace
25 }

27 Component C4 {
28   Version 1
29   URL "http://test.fr/plopC4.jar"
30   DeploymentInterface fr.enac.plop.DIimpl
31 }

33 Component C5 {
34   Version 5
35   URL "http://test.fr/plopC5.jar"
36   Constraint HDSize
37   InitOnly Constraint CpuNet
38 }

40 Component C6 {
41   Version 1
42   URL "http://test.fr/plopC6.jar"
43 }
```

Listing 1: **Component** definition in MuScADeL.

2) Probe: The keyword **Probe** defines a probe (cf. Listing 2). A probe has two fields. The first one, the **ProbeInterface**, specifies the interface of the probe. This interface is needed for interactions with the deployment system for information retrieval. The second one, the **URL**, specifies the address where the probe is reachable for download.

```
1  Probe Arduino {
2    ProbeInterface fr.irit.arduino.DIimpl
3    URL "http://irit.fr/INCOME/arduinoProbe.jar"
4  }
```

Listing 2: **Probe** definition in MuScADeL.

3) BCriterion: The keyword **BCriterion** defines a criterion (cf. Listing 3). A criterion is a conjunction of conditions concerning probed values, like in CpuNet (Listing 3, line 14). There are two kinds of conditions concerning either the existence or liveliness of a probe, or a specific value given by a probe. In the first case, the condition is composed by the probe name and the keywords **Exists** or **Active**, which are defined for any probe interface. For example, in Listing 3, at line 2 and 3,

the used probe is `Arduino`, and conditions use default methods **`Exists`** and **`Active`**. In the second case, the condition is composed by the probe name, the method to call, a comparator, and a value. In this case, the method is probe-specific, and defined in the probe interface. For example, in Listing 3 at line 11, the used probe is `RAM`, the information method used is `freeSpace`, and its value is compared to the number `40`, for 40Mb. A criterion can be used to define both a component constraint (cf. Listing 3, line 37) or a deployment requirement (cf. Listing 5, line 4).

```
1  BCriterion ActiveArduino {
2    Arduino Exists;
3    Arduino Active;
4  }

6  BCriterion LinuxCrit {
7    OS.name = "Linux";            //OS probe
8  }

10 BCriterion Freespace {
11   RAM.freeSpace >= 40;          //RAM probe
12 }

14 BCriterion CpuNet {
15   CPU.load < 80;                //CPU probe
16   Network.bandWith > 1024;      //Network probe
17 }

19 BCriterion HDSize {
20   HD.size > 100;                //HD probe
21 }
```

Listing 3: **`BCriterion`** definition in MuScADeL.

4) Multiscale Probe: The keyword **`MultiScaleProbe`** defines a multiscale probe, useful for deployment requirements (cf. Listing 4). Like **`Probe`**, it has only two fields: **`MultiScaleProbeInterface`** and **`URL`**. A specific keyword is necessary because basic and multiscale probes are considered in a different way when generating the deployment plan. At runtime, a multiscale probe allows to identify the scale or the scale instance of their host device in a given viewpoint/dimension/measure.

```
1 MultiScaleProbe Geography {
2   MultiScaleProbeInterface
3     eu.telecom-sudparis.GeographyProbeImpl
4   URL "http://it-sudparis.eu/INCOME/GeoProbe.jar"
5 }
```

Listing 4: **`MultiScaleProbe`** definition in MuScADeL.

5) Deployment: The keyword **`Deployment`** defines the deployment requirements (cf. Listing 5). The keyword **`AllHosts`** allows to specify and delimit the deployment domain: line 2 expresses that the deployment covers all hosts which satisfy the basic criterion `LinuxCrit`. The operator `@` allows to specify deployment requirement specific to a component. These requirements can take several forms:

- The device hosting the component `C1` must satisfy `CpuNet` and be on the scale `Device.StorageCapacity.Giga` (line 4);
- the component `C2` must be deployed on `2` to `4` devices, in the city `Toulouse` (line 5);
- the component `C3` (line 6) must be deployed on one device (implicit) which has the same value in the dimension `Device.Type` as the device hosting `C1`;
- the component `C4` must be deployed on all devices of the scale `Device.Type.Smartphone`, *i.e.*, on all smartphones of the domain (line 7);
- the component `C5` must be deployed on a device which is situated in the same medium area network (MAN) as the device hosting `C4`, the ratio expression `1/3` specifying that there should be one instance of the component `C5` deployed for three instances of the component `C4` (line 8);
- one instance of the component `C6` must be deployed on each scale instance of the scale `Geography.location.City` (line 9).

```
1  Deployment {
2    AllHosts LinuxCrit;

4    C1 @ CpuNet, Device.StorageCapacity.Giga;
5    C2 @ 2..4, Geography.Location.City("Toulouse");
6    C3 @ SameValue Device.Type(C1);
7    C4 @ All, Device.Type.SmartPhone;
8    C5 @ 1/3 C4, SameValue Network.Type.MAN(C4);
9    C6 @ Each Geography.Location.City;
10 }
```

Listing 5: **`Deployment`** definition in MuScADeL.

The keyword **`DifferentValue`** allows to specify the contrary of **`SameValue`**. Using these keywords, it is possible to define a requirement related to a scale or a scale instance.

6) Dynamics and openness: Some constructions of the DSL are particularly well-adapted for the expression of properties related to dynamics and openness. By default, the properties should be satisfied during the entire application runtime, and so must be checked dynamically. The keyword **`InitOnly`** is used to specify that a constraint should be satisfied initially by the generated deployment plan, but maybe not satisfied at runtime. When specifying deployment requirements, the keyword **`All`** allows to specify that a component should be deployed on a subdomain which satisfies (even dynamically) a requirement. In the example, the component `C4` should be deployed on every smartphone of the domain, including those which enter in the domain at runtime; so, the deployment plan evolves dynamically depending on entering and leaving devices.

As the code can be split in several files, the keyword **`Include`** permits to include other files. One of these file must contain the expression of the deployment.

B. Implementation

Using *Xtext* and *Xtend* frameworks (for lexical and syntactic analysis, translation, and generation of Java source code) [19], we have realized an Eclipse plugin for

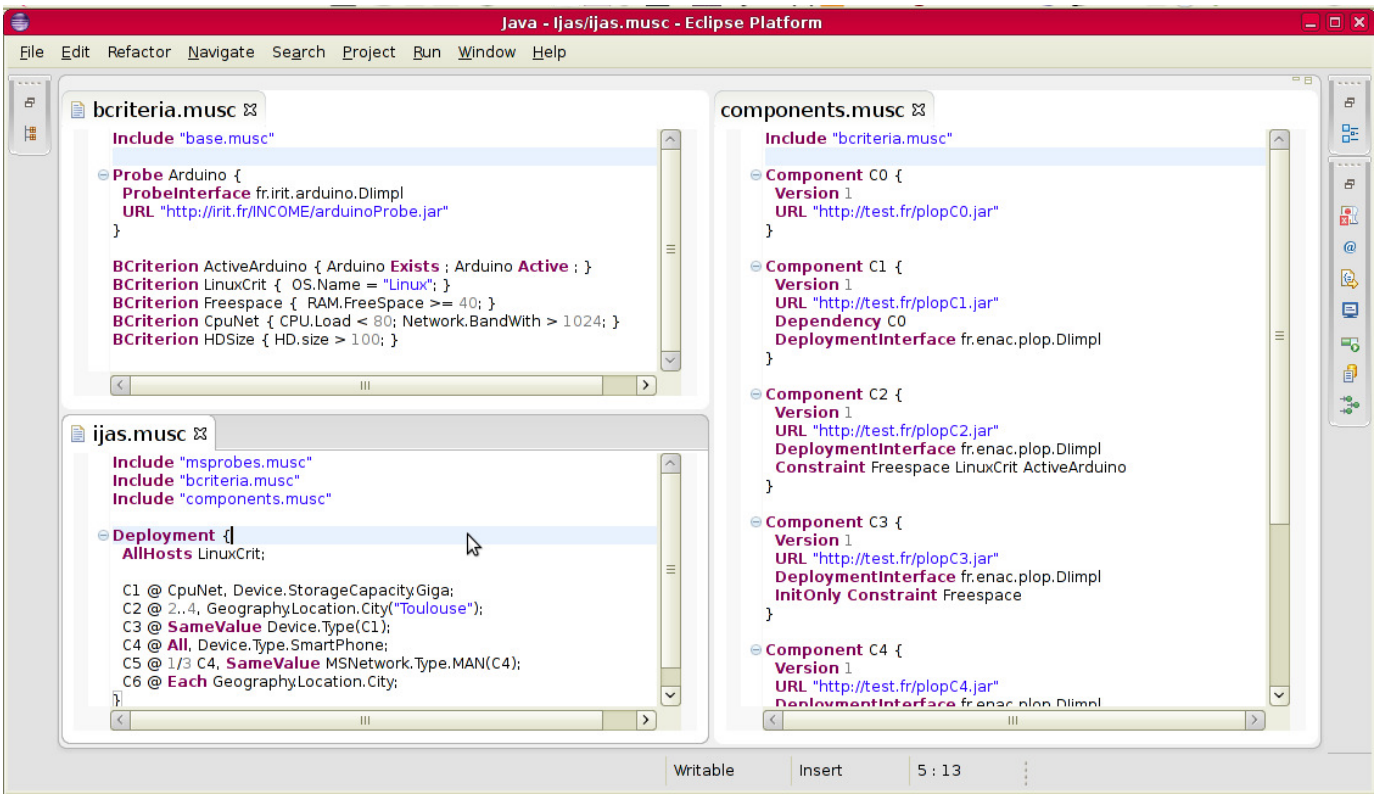


Fig. 5: MuScADeL editor.

the edition of MuScADeL. Using Java and Eclipse makes MuScADeL editor multi-platform compliant and easy-to-use for the deployment designer. Moreover, it runs alongside MuScA (***Mu**ltiscale distributed systems **Sc**ale **A**wareness framework*), a multiscale characterization process, allowing the deployment designer to be able within the same engineering tool to define new multiscale viewpoints, dimensions or scales, before using them in MuScADeL. This binding allows MuScADeL editor to propose an autocompletion of multiscale dimensions and scales and a check of their use. A screenshot of the MuScADeL plugin on Eclipse is shown in Figure 5.

V. From deployment design to a deployment plan

The deployment designer describes the deployment properties using MuScADeL, and then the deployment operator runs the generation of the deployment plan. This generation is a complex process, done in several steps. It is described with a SPEM-like process diagram in Figure 6.

Firstly, the MuScADeL code is compiled, giving in result a file containing the set of components properties, and a file containing a list of probes. The probes are pieces of software that can gather information about a device, such as those described with the *Bcriterion* idiom in the DSL (available memory, OS...). They must be deployed on each host before the deployment, this step being one of the pre-activation activity described in Figure 3.

To achieve the deployment of these probes, we use a light program called *bootstrap*, already installed on each host of the deployment domain. It contains a set of common probes (called later basic probes), and can dynamically acquire and run any other probe specified during this step. Then, the probes are invoked on each device and the results are sent back to the deployment management system.

The set of information gathered on each device is called the *domain state*, representing at that time a view of the status of each device in the deployment domain.

Finally, the constraint solver takes this domain state and the set of properties, and compute a deployment plan as output. Note that we are looking for the first available solution, not to optimize in any way the deployment plan. In case of the constraint solver can not find a solution, the deployment operator is notified and the deployment designer has to change his code.

This process, from MuScADeL edition to a generated deployment plan, is illustrated in a demonstration movie available at http://anr-income.fr/T5/MuScADeL_IDE_Deployment_Plan_Generation.mkv.

The following sections will describe some of the software elements needed to complete the deployment plan, such as the bootstrap architecture and solving step.

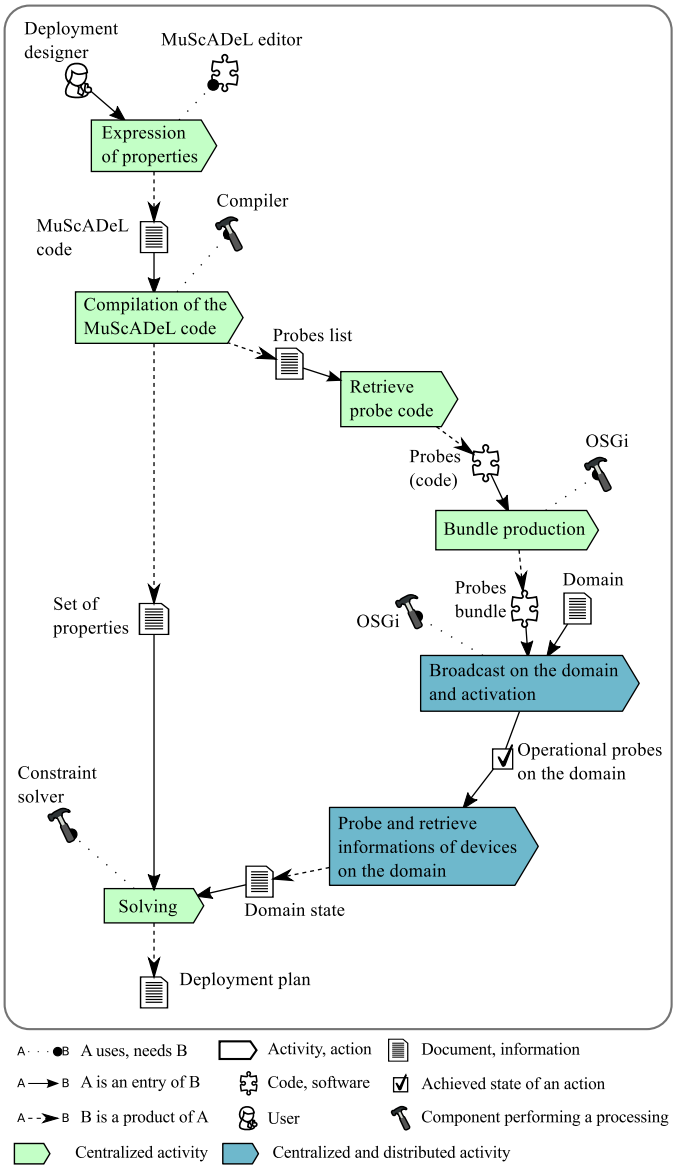


Fig. 6: Generation of the deployment plan.

VI. Bootstrap architecture

In this section, we present the bootstrap, its architecture, and the deployment management system interface (GUI).

A. Bootstrap

The bootstrap of the Deployment Management System (DMS) is a small program available on all devices belonging to the deployment domain. The bootstrap is an OSGi framework containing four bundles: Main, RabbitMQ Client, Basic Probes, and WebService DMS (cf. Figure 7). The Main bundle is the entry point of the bootstrap, and contains the core features of the bootstrap. The RabbitMQ Client bundle insures the link between a device and the deployment monitor. The deployment monitor is a centralized component, in charge of the initial deployment, *e.g.*, probe sending, solving steps, etc., and allows the deployment operator to interact with the DMS. The RabbitMQ Client is useful for the detection of devices appearance and disappearance. The Basic Probes bundle is a set of basic probes, the most common ones. The WebService DMS bundle allows the bootstrap to communicate directly with the deployment monitor. For specific needs, bundles are added to the bootstrap. This extension turns the bootstrap into the device local entity of the DMS.

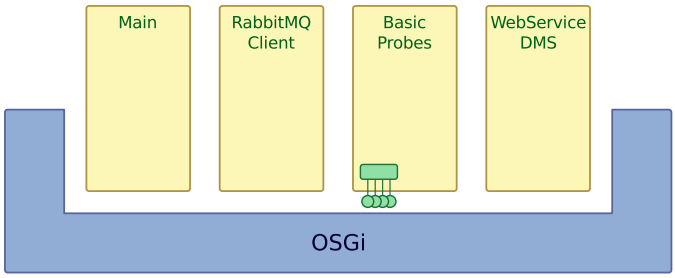


Fig. 7: Bootstrap architecture.

B. Main

The main bundle is the entry point of the bootstrap. It uses the RabbitMQ Client to offer a presence indicator system, and it permits an asynchronous communication system with devices over the network (through firewalls, router, etc.). In this way, it is possible to remotely install, activate, or stop a bundle, or ask for devices state by activating the basic probes service.

C. Basic probes

The Basic Probes bundle contains seven probes, the most useful ones:

- CPU: processor frequency, etc.;
- RAM: free RAM available, full capacity of the RAM, etc.;
- hard disk: full capacity, space available, etc.;
- OS: the operating system, the version of the operating system, etc.;
- network: IP, type of the connection (*e.g.*, Ethernet, WiFi, 3G), etc.;
- and the locale.

Once the deployment monitor sends a request for information, this bundle sends back a packet, on JSON format, containing the result of the probing. An example of returned packet is shown in Listing 6.

D. Android and J2SE

There is two versions of the bootstrap, on in standard Java (J2SE), and another for Android. They are distributed on their native format: *jar* for the J2SE version, and *apk* for the Android version. They contain a start of the OSGi implementation, and the control of the execution. It is the only part of the deployment framework which is not heterogeneous.

```
{
 "basicStatus": {
  "IP_GATEWAY": "81.252.230.88",
  "JAVA_VENDOR": "Oracle Corporation",
  "INTERFACE_LIST": [
   {"name": "wlan0",
    "isUp": true,
    "isLoop": false,
    "ipList": [
     "fe80:0:0:0:e206:e6ff:fecd:4b52%3",
     "10.0.1.146"
    ]},
   {"name": "eth0",
    "isUp": true,
    "isLoop": false,
    "ipList": [
     "fe80:0:0:0:3e97:eff:fe5e:c0db%2",
     "10.0.0.119"
    ]}
  ],
  "CPU_AVAILABLE_PROC": 4,
  "MEM_TOTAL": 115,
  "OS_NAME": "Linux",
  "HD_TOTALSPACE": 179949,
  "MEM_FREE": 92,
  "LOCALE_LANG": "francais",
  "JAVA_NAME": "Java Platform API Specification",
  "HD_FREESPACE": 91082,
  "OS_ARCH": "amd64",
  "OS_VERSION": "3.8.0-34-generic",
  "CPU_NAME": "Intel(R) Core(TM)
               i5-3210M CPU@2.50GHz",
  "JAVA_VERSION": "1.7",
  "JVM_NAME": "Java HotSpot(TM)
               64-Bit Server VM",
  "CPU_SPEED": 1200,
  "LOCALE_COUNTRY": "France",
  "CPU_LOAD": 0.08
 }
}
```


Listing 6: Example of packet returned by Basic Probes bundle on JSON format.

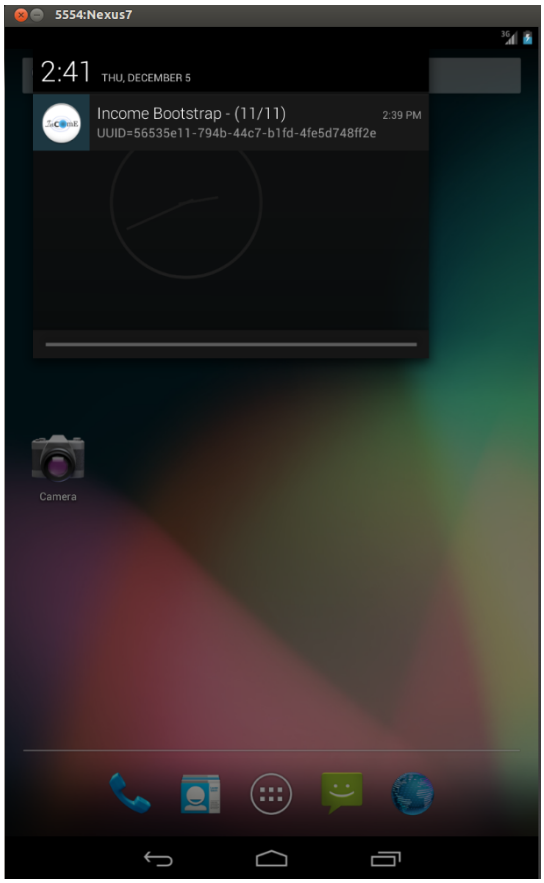


Fig. 8: Android bootstrap - Notification bar.

In both cases, the bootstrap uses the system to indicate to the user that the framework is on. An icon is shown on the bar task, or the notification bar for Android, which allows the user to check the state of the deployment system, stop it, etc. Figure 8 and Figure 9 show pictures of the Android bootstrap. In Figure 8, the notification bar shows the bootstrap icon, and in Figure 9, the bootstrap application interface shows information about installed bundles (full bundles and their status, 32 is for active bundle).

E. DMSMaster

The DMSMaster is a Java software which shows the list of reachable devices in real time, and can ask them to send information about their state (resulting from basic probes). It is a preliminary draft of the final deployment software GUI.

Figure 10 shows two devices running a bootstrap: a 3G connected smartphone and a WiFi connected tablet. The DMSMaster (red circled) lists these two devices and retrieved information.

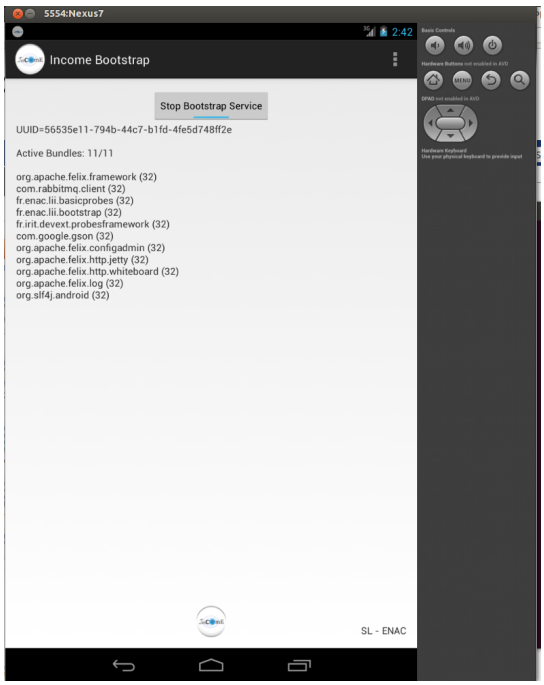


Fig. 9: Android bootstrap interface.

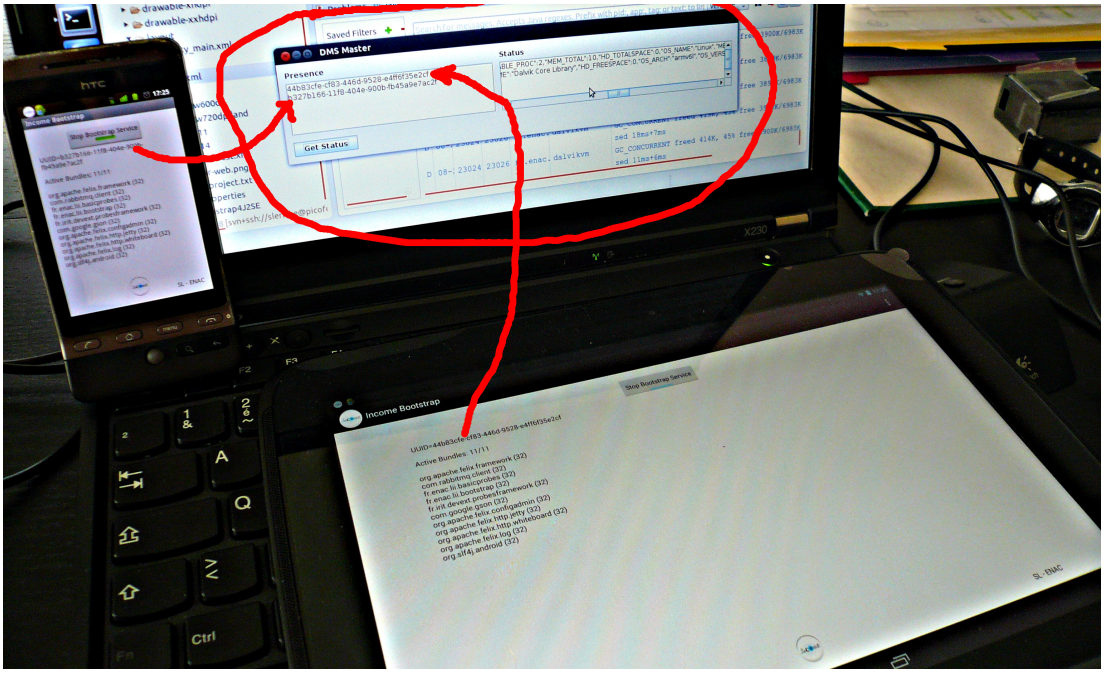


Fig. 10: Picture of the DMSMaster.

VII. Constraints formalization

In this section, we present the formalization of deployment properties.

A. Data and data structure

1) Input data: Analysis of the MuScADeL code allows to identify some properties that the system have to hold. This analysis produces a $Component \times Property$ matrix named $Comp$, defined by

- $Comp(i,j) = 1$ if the deployment of the component $Component_i$ is constrained by the property $Property_j$,
- $Comp(i,j) = 0$ otherwise.

Note that components that are specified to be required for the deployment of another component are taken into account and integrated to the matrix $Comp$. Only simple properties are taken into account for the calculation of $Comp$. Simple properties are basic criteria and multiscale criteria that concern only one component.

On the other hand, independently from the MuScADeL code, analysis of the deployment domain produces a $Device \times Property$ matrix named Dom, defined by

- $Dom(i,j) = 1$ if the device $Device_i$ has property $Property_j$,
- $Dom(i,j) = 0$ otherwise.

Basic and multiscale probes are used to produce the matrix Dom. By and large, measures of devices that are taken by the probes are part of the domain state. They are supplied as an array associating devices and measures.

2) Output data: The deployment plan produced by the solver is a $Component \times Device$ obligation matrix named $Oblig$, defining the placement of each component. It is defined by

- $Oblig(i,j) = 1$ if component $Component_i$ must be deployed on device $Device_j$,
- $Oblig(i,j) = 0$ otherwise.

B. Deployment properties

1) Constraints and requirements: A $Component \times Device$ type possibility matrix named $SatVar$, is build. Each coefficient of $SatVar$ is a variable which can take its value in $\{0,1\}$.

Constraints are added on some coefficients from matrices $Comp$ and Dom. Those constraints are the assignment of the coefficient to the value 0. This assignment correspond to the impossibility for the device to host the component. It is expressed using the following constraint[1] (nb_dev and nb_comp respectively correspond to the number of devices and to the number of components involved in the deployment):

$$\forall i \in \{1,..,nb\_comp\}, \forall j \in \{1,..,nb\_dev\}$$
$$Comp(i) \cdot Dom(j) = \vec{0} \implies SatVar(i,j) = 0 \quad (1)$$

In this formula, $Comp(i)$ and $Dom(j)$ respectively represent rows i and j of matrices $Comp$ and Dom, the operator $\cdot$ constructs a vector composed by the two by two element product of the two given lines, and $\vec{0}$ represents the null vector.

[1]By convention, indexes of row and column matrices and of arrays begin at 1.

2) Number of component instances:

a) Cardinality: For each component (a row of matrix *SatVar*), the number of component instances is the sum of the row's elements. A constraint is then built for each component depending the number of instances.

Thus, if component C_k must be deployed on n_k devices, it would be translated using constraint:

$$\sum_{j=1}^{nb\_dev} SatVar(k,j) = n_k \quad (2)$$

If component C_k must be deployed on n_k to m_k devices, it would be translated using constraint:

$$n_k \leq \sum_{j=1}^{nb\_dev} SatVar(k,j) \leq m_k \quad (3)$$

b) All: The All cardinality specifies that a component must be deployed on all devices that can host it. The number of these devices should be maximized. The expression `Ck @ All` would be translated using following formula:

$$\max_{\substack{j\in\{1,..,nb\_dev\} \\ SatVar(k,j)}} \left(\sum_{i=1}^{nb\_dev} SatVar(k,i) \right) \quad (4)$$

c) Ratio: A ratio between instances of different components can be translated using the same principle than simple cardinality, associating several rows of *SatVar*. The expression `Ck @ n/m Cl` would be translated using constraint:

$$\sum_{i=1}^{nb\_comp} SatVar(k,i) = n \times \left\lfloor \frac{\sum_{i=1}^{nb\_comp} SatVar(l,i)}{m} \right\rfloor \quad (5)$$

where $\lfloor \cdot \rfloor$ refers to floor function.

d) Dependency: When the deployment designer defines a component, he can specify if a component requires another one, by means of keyword **`Dependency`**. In this case, these two components must be on the same device. Suppose component C_k requires component C_l, this would be translated using following formula:

$$\forall i \in \{1,..,nb\_comp\} \quad SatVar(k,i) = 1 \implies SatVar(l,i) = 1 \quad (6)$$

3) Multiscale properties:

a) Dependant components: Multiscale properties expressed by means of the keywords **`SameValue`** and **`DifferentValue`** are defined on several component. Those properties express required conditions for the deployment of components, and are directed by the values provided by the referred multiscale probe. For example, the expression `Ck @ SameValue Some.MS.Scale(Cl)` expresses that instances of components C_k and C_l must be on the same scale instance of `Some.MS.Scale`. Let *MSProbe* be an array which associates for each device the measure of the multiscale probe. It expresses that C_k and C_l are respectively deployed on D_i and D_j only if D_i and D_j have the same value on *MSProbe*, which is:

$$\forall i,j \in \{1,..,nb\_dev\} \quad (SatVar(k,i) \wedge SatVar(l,j)) \implies (MSProbe(i) = MSProbe(j)) \quad (7)$$

b) Placement by scale instance: Finally, a component instance presence on a given scale (expressed by means of the keyword **`Each`**) is defined by a constraint similar to the previous. The set of available devices is limited and identified from measured values by the referred multiscale probe. For example, the expression `Ck @ Each Some.MS.Scale` expresses that one instance of the component C_k must be deployed on each scale instance of `Some.MS.Scale`. In order to do that, two array are required: *MSprobe*, which associates to each device the measure of the required multiscale probe, and *MSProbeId*, which lists unique identifiers of each scale instance. Previous expression would be translated using the constraint (*nb_inst* refers to the number of scale instance):

$$\forall i \in \{1,..,nb\_inst\} \quad \left(\sum_{\substack{j\in\{1,..,nb\_dev\} \\ MSProbeId(i)=MSProbe(j)}} SatVar(k,j) \right) = 1 \quad (8)$$

VIII. MuScADeL Solving

In this section, we present the application of our formalization through the MuScADeL code given in Section IV. Thereafter we present our choice of the constraint solver. Finally we present our library of constraint formalization and its use.

A. Matrices definition

Table Ia gives an exemple of the matrix *Comp* built from the MuScADeL code presented in Section IV. Table Ib gives an example of a matrix *Dom* extracted from the domain state. In these matrices, properties P_1, P_2, P_3, P_4, P_5 et P_6 respectively refer to criteria `CpuNet`, `HDSize`, `Freespace`, `LinuxCrit`, `ActiveArduino`, `Device.Type.Smartphone`, `Device.StorageCapacity.Giga`, and `Geography.Location.City("Toulouse")`.

Note that basic and multiscale probes are used to build *Dom* matrix. Generally, probed measures from devices are part of the domain state. They are provided as an

TABLE I: Data on components and devices.

(a) Component matrix *Comp*.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
C_0	1	0	0	1	0	0	1	0
C_1	1	0	0	1	0	0	1	0
C_2	0	0	1	1	1	0	0	1
C_3	0	0	1	1	0	0	0	0
C_4	0	0	0	1	0	1	0	0
C_5	1	1	0	1	0	0	0	0
C_6	0	0	0	1	0	0	0	0

(b) Device matrix *Dom*.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
D_1	1	1	1	1	0	1	0	0
D_2	1	1	1	1	0	1	0	0
D_3	1	1	1	1	0	1	0	0
D_4	1	1	1	1	0	1	0	1
D_5	1	1	1	1	0	0	1	0
D_6	0	1	0	1	0	0	1	0
D_7	0	1	1	1	0	0	1	0
D_8	1	1	1	1	1	0	0	1
D_9	1	1	1	1	1	0	0	1
D_{10}	1	1	1	1	1	0	0	1
D_{11}	1	1	1	0	1	0	0	1
D_{12}	1	1	1	1	0	1	0	1

TABLE II: Probed data from multiscale probes.

(a) Probed data from `Device`.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}
`Device.Type` Scale	Smartphone	Smartphone	Smartphone	Smartphone	Server	Server	Server	PC	PC	PC	PC	Smartphone

(b) Probed data from `MSNetwork`.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}
`MAN`	betelgeuse	betelgeuse	betelgeuse	persee	orion	orion	orion	persee	persee	persee	persee	persee

(c) Probed data from `Geography`.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}
`City`	Tournefeuille	Tournefeuille	Tournefeuille	Toulouse	Paris	Paris	Paris	Toulouse	Toulouse	Toulouse	Toulouse	Toulouse

TABLE III: Obligation matrix *Oblig*.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}
C_0	0	0	0	0	1	0	0	0	0	0	0	0
C_1	0	0	0	0	1	0	0	0	0	0	0	0
C_2	0	0	0	0	0	0	0	1	1	1	0	0
C_3	0	0	0	0	0	0	1	0	0	0	0	0
C_4	1	1	1	0	0	0	0	0	0	0	0	0
C_5	0	0	1	0	0	0	0	0	0	0	0	0
C_6	0	0	1	0	0	0	1	0	0	0	0	1

array associating the device to the measure. For this example, the solving needs information on device type, network identification, and geolocation. The probing is performed respectively by probes `Device`, `MSNetwork`, and `Geography`: probe `Device` identifies the type of the device using the `Type` dimension, probe `MSNetwork` determinate in which medium area network the device belongs to using the scale `Type.MAN`, and probe `Geography` locate in which city is the device using the scale `Location.City`. They produce respectively Tables IIa, IIb, and IIc.

Table III presents a possible obligation matrix, *i.e.*, a deployment plan for the MuScADeL code given in Section IV, probed data from multiscale probes `Device` (cf. Table IIa), `MSNetwork` (cf. Table IIb), and `Geography` (cf. Table IIc).

B. Constraint solver

For the generation of the deployment plan, a constraint solver is used. We had to make a choice on which one to use. Table IV depicts a comparison of constraint solvers. We choose for this comparison: Cream [20], Copris [21], JaCoP [22], or-tools [23], jOpt [24], and Choco [25]. All of them are Java compatible, either written in Java, either can be interfaced with Java. There are different kinds of problem that are handled by constraint programming. Constraint solvers are specialized on several kinds of problems, because their solving is treated differently. In Table IV, acronyms CSP, COP, CP, and JS are respectively constraint satisfaction problem, constraint optimization problem, constraint problem, and job scheduling. We are not specialized on constraint solving problem, and look for a constraint solver easy to use. We compare constraint solvers according the

kinds of problem handled, if they are maintained or deprecated, and if the documentation is up-to-date, and helpful. We do not compare their resolution time because the generation of deployment plan is not crucial on time.

TABLE IV: Constraint solvers comparison.

	Problem	Maintenance	Documentation
Cream	CSP	deprecated (2008)	light
Copris	COP,CSP,CP	maintained	almost nonexistent
JaCoP	CSP	maintained	existent
or-tools	CSP	maintained	almost nonexistent
jOpt	CSP,JS	maintained	missing
Choco	CSP	maintained	complete

As our problem is a constraint satisfaction problem, the kind of problem is not discriminating. The most pertinent for us is **Choco**. The library is simple to use, with the most complete documentation.

C. MuScADeLSolving library

We present here the library MuScADeLSolving. It is composed by the class `MuscadelSolving` (listing 8), its interface `MuscadelSolvingInter` (listing 7), and an exception class `MuscadelSolvingExc`.

The interface `MuscadelSolvingInter` contains methods for constraint addition: `simpleCardinality`, `intervalCardinality`, `allCardinality`, `ratio`, `sameDevice`, `sameValue`, `differentValue`, and `each`[2]. Method `solving` launches the solving of the problem.

```
1  public interface MuscadelSolvingInter {
2      public void simpleCardinality (int cmp,
3                      int card);
4      public void intervalCardinality(int cmp,
5                      int min, int max);
6      public void allCardinality (int cmp);
7      public void ratio(int cmpP, int cmpS,
8                      int ratioP, int ratioS);
9      public void sameDevice(int cmp, int dependsOn);
10     public void sameValue(int cmp1, int cmp2,
11                     String[] probedData);
12     public void differentValue(int cmp1, int cmp2,
13                     String[] probedData);
14     public void each (int cmp, String[] probedData);
15     public int[][] solving()
16                     throws MuscadelSolvingExc;
17 }
```

Listing 7: Interface `MuscadelSolvingInter`.

The class `MuscadelSolving` contains matrices *Comp* and *Dom*, the Choco model and the possibility matrix *SatVar*. *SatVar* is built at the creation of an object `MuscadelSolving` (the constructor calls method `preprocessing`).

```
1  public class MuscadelSolving
2                  implements MuscadelSolvingInter{
3    private Model model;
4    private IntegerVariable[][] satVar;
5    private int nb_comp, nb_app nb_prop;
6    private int[][] comp, dom;
7    private ArrayList<Integer> toMaximize;

9    public MuscadelSolving (int[][] comp, int[][] dom) {
10     assert(comp.length > 0) :
11         "MuscadelSolving:␣empty␣comp";

13     this.model = new CPModel();
14     this.nb_comp = comp.length;
15     this.nb_app = dom.length;
16     this.nb_prop = comp[0].length;
17     this.satVar = new IntegerVariable[nb_comp][nb_app];
18     this.comp = comp;
19     this.dom = dom;
20     toMaximize = new ArrayList<Integer>();

22     preprocessing();
23   }
```

Listing 8: Class `MuscadelSolving`.

Method `preprocessing` builds the possibility matrix `satVar` and adds to it constraints related to the impossibility for the device to host the component, as described by formula (1).

```
1  private void preprocessing () {
2    int [] buffer = new int[nb_prop];
3    // For each variable the domain is defined
4    int[] values = {0,1};
5    for (int i = 0; i < nb_comp; i++) {
6      for (int j = 0; j < nb_app; j++) {
7        satVar[i][j] =
8          Choco.makeIntVar("var_" + i + "_" + j, values );
9        model.addVariable(satVar[i][j]);
10     }
11   }
12   for (int i = 0; i < nb_comp; i++) {
13     for (int j = 0; j < nb_app; j++) {
14       boolean cont = true;
15       for (int k = 0; k < nb_prop; k++) {
16         if (!cont) break;
17         buffer[k] = comp[i][k] * dom[j][k];
18         cont = cont & (buffer[k] == comp[i][k]);
19       }
20       if (!cont)
21         model.addConstraint(Choco.eq(0, satVar[i][j]));
22     }
23   }
24 }
```

Listing 9: Method `MuscadelSolving.preprocessing`.

Method `simpleCardinality` adds simple cardinality constraint, *e.g.*, as described by the formula (2).

```
1  public void simpleCardinality (int cmp, int card) {
2    model.addConstraint(Choco.eq(card,
3                          Choco.sum(satVar[cmp])));
4  }
```

Listing 10: Method `MuscadelSolving.simpleCardinality`.

Method `intervalCardinality` adds interval cardinality constraints –*e.g.*, in listing 5 at line 5– as described by the formula (3). In addition to constraints, this method adds the row of the given component to the list of `satVar` rows to maximize.

```
1  public void intervalCardinality(int cmp, int min, int max) {
2    model.addConstraint(Choco.leq(min,
3                          Choco.sum(satVar[cmp])));
4    model.addConstraint(Choco.geq(max,
5                          Choco.sum(satVar[cmp])));
6    toMaximize.add(cmp);
7  }
```

Listing 11: Method `MuscadelSolving.intervalCardinality`.

[2]In the Java code, indexes of row and column matrices and of arrays begin at 0.

Method `allCardinality` adds `All` cardinality constraint, as described by the formula (4). A constraint is added to specify that at least one instance of a component must be deployed on the deployment domain, and the row corresponding to the component in matrix `satVar` is added to the list of rows to maximize.

```
1 public void allCardinality (int cmp) {
2   model.addConstraint(Choco.leq(1,
3                       Choco.sum(satVar[cmp])));
4   toMaximize.add(cmp);
5 }
```

Listing 12: Method `MuscadelSolving.allCardinality`.

Method `ratio` adds a ratio constraint between components, as described by the formula (5). It has as parameters the components concerned by the ratio, the numerator and the denominator.

```
1 public void  ratio (int cnum, int cdenom,
2                     int rnum, int rdenom) {
3   Constraint ratio =
4     Choco.eq(Choco.sum(satVar[cnum]),
5       Choco.mult(rnum,
6         Choco.div(Choco.sum(satVar[cdenom]), rdenom)));
7   model.addConstraint(ratio);
8 }
```

Listing 13: Method `MuscadelSolving.ratio`.

Method `sameValue` and `differentValue` add multiscale dependant component constraints, as described by the formula (7). They have as parameters referred components and an array of probed data, *e.g.*, the array in Table IIc.

```
1  public void sameValue (int cmp1, int cmp2,
2                         String[] probedData) {
3    checkValue(cmp1, cmp2, probedData, true);
4  }
5  public void differentValue(int cmp1, int cmp2,
6                             String[] probedData) {
7    checkValue(cmp1, cmp2, probedData, false);
8  }
9  private void checkValue (int cmp1, int cmp2,
10                  String[] probedData, boolean diff) {
11   assert probedData.length == nb_app :
12         "checkValue␣tab␣size␣problem␣!";

14   for (int m1 = 0; m1 < nb_app; m1++) {
15     for (int m2 = 0; m2 < nb_app; m2++) {
16       if (! (diff ^ probedData[m1].
17                 equals(probedData[m2])))
18           continue;
19       model.addConstraint(Choco.geq(1,
20         Choco.plus(satVar[cmp1][m1],
21             satVar[cmp2][m2])));
22     }
23   }
24 }
```

Listing 14: Methods `MuscadelSolving.sameValue` and `MuscadelSolving.differentValue`.

Method `each` adds constraint related to the placement of a component instance by scale instance, as described by the formula (8). It has as parameter the referred component and an array of probed data.

```
1  public void each (int cmp, String[] probedData) {
2    assert probedData.length == nb_app :
3          "Each␣tab␣size␣problem␣!";

5    HashMap<String, ArrayList<Integer>> id =
6      new HashMap<String,ArrayList<Integer>>();
7    // Construction of id/index map
8    for (int i = 0; i < probedData.length; i++) {
9      if (id.containsKey(probedData[i]))
10       id.get(probedData[i]).add(i);
11     else {
12       ArrayList<Integer> ids = new ArrayList<Integer>();
13       ids.add(i);
14       id.put(probedData[i], ids);
15     }
16   }
17   // Constraint addition
18   for (String str : id.keySet()) {
19     IntegerExpressionVariable add=Choco.ZERO;
20     for (Integer index : id.get(str)) {
21       add = Choco.plus(satVar[cmp][index], add);
22     }
23     Constraint check = Choco.eq(1, add);
24     model.addConstraint(check);
25   }
26 }
```

Listing 15: Method `MuscadelSolving.each`.

Method `sameDevice` adds constraint related to component dependency, as described by the formula (6). This dependency is specified at the component definition, as shown in the listing 1 at line 9.

```
1 public void sameDevice (int cmp, int dependsOn) {
2   Constraint[] ors = new Constraint[nb_app];
3   for (int i = 0; i < nb_app; i++) {
4     ors[i] = Choco.eq(2,
5       Choco.plus(satVar[cmp][i], satVar[dependsOn][i]));
6   }
7   model.addConstraint(Choco.or(ors));
8 }
```

Listing 16: Method `MuscadelSolving.sameDevice`.

Method `solving` launches the constraint solver's solving. If there is no row on the matrix `satVar` to maximize, the solving is launched directly. Otherwise, maximization instructions are added to the Choco model, then the solving is launched. Thereafter, the feasibility of the problem is checked: if the problem has no solution an exception `MuscadelSolvingExc` is thrown, otherwise, the first solution is returned.

```
1  public int[][] solving() throws MuscadelSolvingExc {
2    Solver solver = new CPSolver();

4    if (toMaximize.size() == 0) {
5      solver.read(model);
6      solver.solve();
7    } else {
8      int up = nb_app*toMaximize.size();
9      IntegerVariable maxx = Choco.makeIntVar("max", 1, up);
10     IntegerExpressionVariable add = Choco.ZERO;
11     for (Integer all : toMaximize) {
12       add = Choco.plus(add, Choco.sum(satVar[all]));
13     }
14     model.addConstraint(Choco.eq(maxx, add));
15     solver.read(model);
16     solver.maximize(solver.getVar(maxx),true);
17   }

19   try{
20     if (solver.isFeasible()) {
21       int [][] result = new int[nb_comp][nb_app];
22       for (int i = 0; i < nb_comp; i++) {
23         for (int j = 0; j < nb_app; j++) {
24           result[i][j] =
25             solver.getVar(satVar[i][j]).getVal();
26         }
27       }
28       return result;
29     } else {
30       throw (new MuscadelSolvingExc("No␣solution")); }
```

```
31    } catch (NullPointerException e) {
32      throw (new MuscadelSolvingExc("No solution"));}
33  }
```

Listing 17: Method `MuscadelSolving.solving`.

D. MuScADeLSolving library use

The class `IJAS` (listing 18) presents the constraint addition phase of the exemple presented in Section IV, listing 5. It contains the main method that builds matrices *Comp* and *Dom* (for the example they are given and not generated by the MuScADeL code analysis), calls MuScADeLSolving methods to add specific constraints, launches the solving, and prints the resulting matrix *Oblig*. The console output is shown in listing 19. This program represents the calculation of the deployment plan of the MuScADeL code presented in Section IV, listing 5.

```
1   public class IJAS {
2     static void printOblig(int[][]oblig) { ... }
3     public static void main(String[] args) {
4       int [][] comp = {
5           { 1, 0, 0, 1, 0, 0, 1, 0 },
6           { 1, 0, 0, 1, 0, 0, 1, 0 },
7           { 0, 0, 1, 1, 1, 0, 0, 1 },
8           { 0, 0, 1, 1, 0, 0, 0, 0 },
9           { 0, 0, 0, 1, 0, 1, 0, 0 },
10          { 1, 1, 0, 1, 0, 0, 0, 0 },
11          { 0, 0, 0, 1, 0, 0, 0, 0 },
12      };
13      int[][] dom = {
14          { 1, 1, 1, 1, 0, 1, 0, 0 },
15          { 1, 1, 1, 1, 0, 1, 0, 0 },
16          { 1, 1, 1, 1, 0, 1, 0, 0 },
17          { 1, 1, 1, 1, 0, 1, 0, 1 },
18          { 1, 1, 1, 1, 0, 0, 1, 0 },
19          { 0, 1, 0, 1, 0, 0, 1, 0 },
20          { 0, 1, 1, 1, 0, 0, 1, 0 },
21          { 1, 1, 1, 1, 1, 0, 0, 1 },
22          { 1, 1, 1, 1, 1, 0, 0, 1 },
23          { 1, 1, 1, 1, 1, 0, 0, 1 },
24          { 1, 1, 1, 0, 1, 0, 0, 1 },
25          { 1, 1, 1, 1, 0, 1, 0, 1 }
26      };

28      System.out.println(
29          "\tGeneration of the deployment plan");
30      MuscadelSolving solv =
31          new MuscadelSolving(comp, dom);

33      //C1 @ CpuNet, Device.StorageCapacity.Giga;
34      solv.simpleCardinality(1, 1);

36      // CO requirements are the same than C1;
37      solv.sameDevice(1,0);
38      solv.simpleCardinality(0, 1);

40      // C2 @ 2..4, Geography.Location.City("Toulouse");
41      solv.intervalCardinality(2, 2, 4);

43      // C3 @ SameValue Device.Type(C1);
44      solv.simpleCardinality(3, 1);
45      String[] deviceType =
46        { "Smartphone", "Smartphone", "Smartphone",
47          "Smartphone", "Server", "Server", "Server",
48          "PC", "PC", "PC", "PC", "Smartphone" };
49      solv.sameValue(3, 1, deviceType);

51      // C4 @ All, Device.Type.Smartphone;
52      solv.allCardinality(4);

54      // C5 @ 1/3 C4, SameValue MSNetwork.Type.MAN(C4);
55      solv.ratio(5, 4, 1, 3);
56      String[] man =
57        { "betelgeuse", "betelgeuse", "betelgeuse",
58          "persee", "orion", "orion", "orion", "persee",
59          "persee", "persee", "persee", "persee" };
60      solv.sameValue(5, 4, man);

62      // C6 @ Each Geography.Location.City;
63      String[] cities =
64        { "Tournefeuille", "Tournefeuille", "Tournefeuille",
65          "Toulouse", "Paris", "Paris", "Paris", "Toulouse",
66          "Toulouse", "Toulouse", "Toulouse", "Toulouse" };
67      solv.each(6, cities);

69      try {
70        int[][] oblig = solv.solving();
71        printOblig(oblig);
72      } catch (MuscadelSolvingExc e) {
73        System.err.println("Problem during the solving.");
74      }
75    }
76  }
```

Listing 18: Main class `IJAS`.

```
1   Generation of the deployment plan
2   Oblig :
3   C0 : 0 0 0 0 1 0 0 0 0 0 0 0
4   C1 : 0 0 0 0 1 0 0 0 0 0 0 0
5   C2 : 0 0 0 0 0 0 0 1 1 1 0 0
6   C3 : 0 0 0 0 0 0 1 0 0 0 0 0
7   C4 : 1 1 1 0 0 0 0 0 0 0 0 0
8   C5 : 0 0 1 0 0 0 0 0 0 0 0 0
9   C6 : 0 0 1 0 0 0 1 0 0 0 0 1
```

Listing 19: Console output.

IX. Conclusion and future work

In this paper, we firstly present MuScADeL, a DSL for multiscale and autonomic deployment, and explain the various elements of the language by means of an example. Then, we present how the deployment plan is computed, using a compiler, and a constraint solver. MuScADeL allows to express the deployment properties of a multiscale software system and its components. These properties drive the computation of the deployment plan, and are used by the autonomic deployment system do detect (and possibly repair) any property violation at the application runtime.

Another part of our work concerns the realization of this autonomic deployment system. We are designing it as a middleware, on the same basis than first experiments described in our previous work [9]. This middleware will enable deployment in multiscale environments. It provides the probes needed to gather information about the hosts.

We believe that a DSL is the best way for a deployment designer to describe deployment requirements and constraints. A DSL has much more expressiveness than any Markup Language (such as XML), and is more efficient since the deployment designer expresses (and read) directly concepts of its field of expertise. Moreover, modern tools for making DSL allows their designers to integrate several level of validation (not only syntactic but also semantic).

Presently, MuScADeL targets the installation and activation activities. Other activities and features, as property infringement at application runtime, are hard coded

in the deployment management system. We plan to move some of them at the DSL level, to increase expressiveness and flexibility when designing deployment. For example, we can add in the grammar the keyword **`on-deinstall`** or **`on-update`** to define actions to perform when deinstalling or updating a component.

Focusing on multiscale systems, we do need a sound and extensible vocabulary to describe the dimensions and their scales. In the INCOME project, another ongoing work aims at defining an ontology for multiscale distributed systems. We continuously integrate these concepts in MuScADeL.

Besides, we are currently working on a toolchain for our DSL. Using Xtext and Xtend frameworks [19], we have realized an Eclipse plugin for the edition of the DSL that makes it multi-platform compliant and easy-to-use for a deployment designer. We have also realized a compiler and a solving algorithm to generate the deployment plan. Using this IDE and the compiler, the deployment designer expresses deployment properties, and launches the generation of the deployment plan. The DSL, the Eclipse plugin, the compiler, and the solving algorithm are part of the deliverables of the INCOME project.

Acknowledgments

This work is part of the French National Research Agency (ANR) project INCOME [5] (ANR-11-INFR-009, 2012-2015). The authors thank all the members of the project that contributed directly or indirectly to this paper.

Appendix

This appendix presents the EBNF syntax of MuScADeL.

⟨*root*⟩ ::= ⟨*muscadel-element*⟩+

⟨*muscadel-element*⟩ ::= ⟨*include*⟩
| ⟨*probe*⟩
| ⟨*bcriterion*⟩
| ⟨*component*⟩
| ⟨*msprobe*⟩
| ⟨*deployment*⟩

⟨*include*⟩ ::= 'Include' '"' ⟨*file-id*⟩ '"'

⟨*probe*⟩ ::= 'Probe' ⟨*probe-id*⟩ '{'
'ProbeInterface' ⟨*interface*⟩
('URL' ⟨*string*⟩)?
'}'

⟨*probe-id*⟩ ::= ⟨*id*⟩

⟨*interface*⟩ ::= ⟨*interface-id*⟩ ('.' ⟨*interface-id*⟩)*

⟨*interface-id*⟩ ::= ⟨*id*⟩

⟨*bcriterion*⟩ ::= 'BCriterion' ⟨*bcriterion-id*⟩ '{'
(⟨*condition*⟩ ';')+
'}'

⟨*bcriterion-id*⟩ ::= ⟨*id*⟩

⟨*condition*⟩ ::= ⟨*probe-id*⟩ '.' ⟨*method-id*⟩ ⟨*comp*⟩
⟨*probe-value*⟩
| ⟨*probe-id*⟩ 'Exists'
| ⟨*probe-id*⟩ 'Active'

⟨*method-id*⟩ ::= ⟨*id*⟩

⟨*probe-value*⟩ = ⟨*int*⟩
| ⟨*string*⟩

⟨*comp*⟩ ::= '<' | '>' | '<=' | '>=' | '='

⟨*component*⟩ ::= 'Component' ⟨*component-id*⟩ '{'
'Version' ⟨*int*⟩
'URL' ⟨*string*⟩
('DeploymentInterface' ⟨*interface*⟩)?
('Dependency' (⟨*component-id*⟩)+)?
('InitOnly'? 'Constraint' ⟨*bcriterion-id*⟩)*
'}'

⟨*component-id*⟩ ::= ⟨*id*⟩

⟨*msprobe*⟩ ::= 'MultiScaleProbe' ⟨*msprobe-id*⟩ '{'
'MultiScaleProbeInterface' ⟨*interface*⟩
'URL' ⟨*string*⟩
'}'

⟨*ms-probe-id*⟩ ::= ⟨*id*⟩

⟨*deployment*⟩ ::= 'Deployment' '{'
('AllHosts' (⟨*bcriterion-id*⟩)+ ';')?
(⟨*deployment-requirement*⟩ ';')+
'}'

⟨*deployment-requirement*⟩ ::= ⟨*component-id*⟩ '@'
⟨*requirement-rhs*⟩ (',' ⟨*requirement-rhs*⟩)+
';'

⟨*requirement-rhs*⟩ ::= 'Each' ⟨*mscriterion-scale*⟩
| 'SameValue' ⟨*mscriterion-dependency*⟩
| 'DifferentValue' ⟨*mscriterion-dependency*⟩
| ⟨*mscriterion*⟩
| ⟨*bcriterion-id*⟩
| ⟨*ratio*⟩
| ⟨*location*⟩
| ⟨*cardinality*⟩

⟨*mscriterion-dependency*⟩ ::= ⟨*msprobe-id*⟩ '.' ⟨*dim-id*⟩ '('
⟨*component-id*⟩ ')'
| ⟨*msprobe-id*⟩ '.' ⟨*dim-id*⟩ '.' ⟨*scale-id*⟩ '('
⟨*component-id*⟩ ')'

⟨*mscriterion-scale*⟩ ::= ⟨*msprobe-id*⟩ '.' ⟨*dim-id*⟩ '.' ⟨*scale-id*⟩

⟨*mscriterion*⟩ ::= ⟨*mscriterion-scale*⟩
| ⟨*msprobe-id*⟩ '.' ⟨*dim-id*⟩ '.' ⟨*scale-id*⟩ '(' ⟨*string*⟩ ')'

⟨*dim-id*⟩ ::= ⟨*id*⟩

⟨*sc-id*⟩ ::= ⟨*id*⟩

⟨*ratio*⟩ ::= ⟨*int*⟩ '/' ⟨*int*⟩ ⟨*component-id*⟩

⟨*location*⟩ ::= ⟨*int*⟩ '.' ⟨*int*⟩ '.' ⟨*int*⟩ '.' ⟨*int*⟩

⟨*cardinality*⟩ ::= ⟨*int*⟩
| ⟨*interval*⟩
| 'All'

⟨*interval*⟩ ::= ⟨*int*⟩ '..' ⟨*int*⟩

References

[1] R. Boujbel, S. Leriche, and J.-P. Arcangeli, "A DSL for multi-scale and autonomic software deployment," in International Conference on Software Engineering Advances (ICSEA 2013), L. Lavazza, R. Oberhauser, A. Martin, J. Hassine, M. Gebhart, and M. Jäntti, Eds., 2013, pp. 291–296.

[2] G. Blair and P. Grace, "Emergent middleware: Tackling the interoperability problem," IEEE Internet Computing, vol. 16, no. 1, pp. 78–82, jan.-feb. 2012.

[3] M. Kessis, C. Roncancio, and A. Lefebvre, "DASIMA: A flexible management middleware in multi-scale contexts," in 6th Int. Conf. on Information Technology: New Generations (ITNG '09), april 2009, pp. 1390–1396.

[4] M. van Steen, G. Pierre, and S. Voulgaris, "Challenges in very large distributed systems," Journal of Internet Services and Applications, vol. 3, no. 1, pp. 59–66, 2012.

[5] J.-P. Arcangeli, A. Bouzeghoub, V. Camps, C. M.-F. Canut, S. Chabridon, D. Conan, T. Desprats, R. Laborde, E. Lavinal, S. Leriche, H. Maurel, A. Péninou, C. Taconet, and P. Zaraté, "INCOME - Multi-scale context management for the Internet of Things," in Ambient Intelligence, 3rd Int. Joint Conf. AmI 2012, ser. Lecture Notes in Computer Science, F. Paternò, B. E. R. d. Ruyter, P. Markopoulos, C. Santoro, E. v. Loenen, and K. Luyten, Eds., vol. 7683. Springer, 2012, pp. 338–347.

[6] S. Rottenberg, S. Leriche, C. Lecocq, and C. Taconet, "Vers une définition d'un système réparti multi-échelle," in Journées francophones Mobilité et Ubiquité (UBIMOB). Cépaduès Editions, 2012, In French.

[7] S. Rottenberg, S. Leriche, C. Taconet, C. Lecocq, and T. Desprats, "From Smartdust to Cloud: The emergence of multiscale distributed systems," 2013, Unpublished Paper.

[8] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. van der Hoek, and A. L. Wolf, "A characterization framework for software deployment technologies," Defense Technical Information Center (DTIC) Document, Tech. Rep., april 1998.

[9] M. E. A. Matougui and S. Leriche, "A middleware architecture for autonomic software deployment," in The 7th Int. Conf. on Systems and Networks Communications (ICSNC'12). Lisbon, Portugal: XPS, 2012, pp. 13–20, 12619 12619. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00755352

[10] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1, pp. 41–50, 2003.

[11] A. Van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography," ACM Sigplan Notices, vol. 35, no. 6, pp. 26–36, 2000.

[12] M. Strembeck and U. Zdun, "An approach for the systematic development of domain-specific languages," Software: Practice and Experience, vol. 39, no. 15, pp. 1253–1292, 2009.

[13] J.-P. Tolvanen and S. Kelly, "Integrating models with domain-specific modeling languages," in Proceedings of the 10th Workshop on Domain-Specific Modeling, ser. DSM '10. New York, NY, USA: ACM, 2010, pp. 10–1. [Online]. Available: 10.1145/2060329.2060354

[14] A. Flissi, J. Dubus, N. Dolet, and P. Merle, "Deploying on the grid with deployware," in CCGRID. IEEE Computer Society, 2008, pp. 177–184.

[15] A. Dearle, G. N. C. Kirby, and A. J. McCarthy, "A framework for constraint-based deployment and autonomic management of distributed applications," in International Conference on Autonomic Computing (ICAC'04). IEEE Computer Society, 2004, pp. 300–301.

[16] A. Dearle, G. N. C. Kirby, and A. McCarthy, "A middleware framework for constraint-based deployment and autonomic management of distributed applications," CoRR, vol. abs/1006.4733, 2010.

[17] K. Sledziewski, B. Bordbar, and R. Anane, "A DSL-based approach to software development and deployment on cloud," in 24th IEEE Int. Conf. on Advanced Information Networking and Applications (AINA 2010). IEEE Computer Society, 2010, pp. 414–421.

[18] S. Malek, N. Medvidovic, and M. Mikic-Rakic, "An extensible framework for improving a distributed software system's deployment architecture," IEEE Transactions on Software Engineering, vol. 38, no. 1, pp. 73–100, 2012.

[19] M. Eysholdt and H. Behrens, "Xtext: implement your language faster than the quick and dirty way," in SPLASH/OOPSLA Companion, ser. Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA, W. R. Cook, S. Clarke, and M. C. Rinard, Eds. ACM, 2010, pp. 307–309.

[20] N. Tamura, "Cream: Class library for constraint programming in Java," last access: June 2014. [Online]. Available: http://bach.istc.kobe-u.ac.jp/cream

[21] ——, "Copris: Constraint Programming in Scala," last access: June 2014. [Online]. Available: http://bach.istc.kobe-u.ac.jp/copris/

[22] K. Kuchcinski and R. Szymanek, "JaCoP - Java constraint programming solver," last access: June 2014. [Online]. Available: http://jacop.osolpro.com/

[23] "or-tools, operations research tools developed at Google," last access: June 2014. [Online]. Available: https://code.google.com/p/or-tools/

[24] "jOpt, Java OPL implementation," last access: June 2014. [Online]. Available: http://jopt.sourceforge.net/opl.php

[25] C.H.O.C.O. Team, "CHOCO: an open source Java constraint programming library," Ecole des Mines de Nantes, Tech. Rep. 10-02-INFO, 2010, last access: June 2014. [Online]. Available: http://www.emn.fr/z-info/choco-solver/

An Overall Framework for Reasoning About UML/OCL Models Based on Constraint Logic Programming and MDA

Beatriz Pérez
Department of Mathematics and Computer Science,
University of La Rioja,
Logroño, Spain.
Email: beatriz.perez@unirioja.es

Ivan Porres
Department of Information Technologies,
Åbo Akademi University,
Turku, Finland
Email: ivan.porres@abo.fi

***Abstract*—Due to the widespread adoption of the Model Driven Engineering paradigm, models have become cornerstone components in the software development process. This fact requires verifying such models' correctness in order to ensure the quality of the final product. In this context, the Unified Modeling Language (UML) and the Object Constraint Language (OCL) constitute two of the most commonly used modeling languages. We have defined an overall framework to reason about UML/OCL models based on Constraint Logic programming (CLP). In particular, as model finding and design space exploration tool, we use Formula. We show how to translate a UML model into a CLP program following a Meta–Object Facility (MOF) like framework. Furthermore, we enhance our proposal by identifying an expressive fragment of OCL, which guarantees finite satisfiability and we show its translation to Formula. We also complete our approach by developing the *CD2Formula* Eclipse plug–in, which implements, following a Model Driven Architecture (MDA) approach, our UML model to Formula translation proposal. The proposed framework can be used to reason, validate and verify UML software designs by checking correctness properties and generating model instances using the model exploration tool Formula.**

Keywords–UML, OCL, Constraint Logic Programming, reasoning, model verification, MDA

I. INTRODUCTION

This paper is an extension of the work presented in [1]. Due to the widespread adoption of the Model Driven Engineering (MDE) [2] paradigm, models have become cornerstone components in the software development process. This fact requires verifying not only the completeness of such models but also their correctness in order to ensure the quality of the final product, reducing time to market and decreasing development costs. In this context, the Unified Modeling Language (UML) and the Object Constraint Language (OCL) constitute two of the most commonly used modeling languages. On the one hand, UML [3] has been widely accepted as the de–facto standard for building object-oriented software. OCL [4], on the other hand, has been introduced into UML as a logic-based sublanguage to express integrity constraints that UML diagrams cannot convey by themselves.

Unfortunately, in some occasions, possible design errors are not detected until the later implementation stages, thus increasing the cost of the development process [5], [6]. This situation requires a wide adoption of formal methods within the software engineering community. In this line, there have been remarkable efforts to formalize UML semantics to solve ambiguity and under specification detected in UML's semantics. The formalization and analysis of the specific UML modeled artifacts can be done by carrying out a translation to another language that preserves its semantics [5], [6], [7], [8]. The resulted translation can be used to reason about the software design by checking predefined correctness properties about the original model [6].

In this paper, we propose to use the *Constraint Logic programming* (CLP) paradigm as a complementary method for UML modeling foundations, including models' satisfiability and inspection. More specifically, we focus on UML class diagrams (CD), annotated with OCL constraints, which are considered to be the mainstay of Object-Oriented analysis and design for representing the static structure of a system. Considering CD/OCL models as model representation, we propose an overall framework to reason about such models based on CLP. In particular, as model finding and design space exploration tool we use Formula [9], which stands on algebraic data types (ADT) and CLP, and which has been proved to provide several advantages, including more expressivity, over using other tools [10], [11]. The defined framework is two–fold. Firstly, we have conceptually defined a proposal for the translation of CD/OCL models to Formula. Secondly, we have used a Model Driven Development (MDA) based approach [12] to automatically generate the Formula specification from a class diagram. As for the first contribution, we give a proposal for the translation of a UML model into a Constraint Satisfaction Problem following a multilevel Meta–Object Facility (MOF) like framework. We enhance our proposal by identifying a fragment of OCL that guarantees finite satisfiability, while being, at the same time, considerably expressive. We also show how to translate such OCL fragment to Formula, by giving, as an intermediate step, a representation of the OCL constraints as First-Order Logic (FOL) expressions. As for the second contribution, we have implemented our class diagram to Formula translation approach by using a model-to-text (M2T) transformation tool, obtaining a set of transformation files defined in such a tool. Additionally, we have integrated the resulted files into an Eclipse plug–in, called *CD2Formula* plug–in, we have developed to easily

and automatically transform a class diagram to Formula. The proposed framework can be used to reason, validate and verify UML software designs by checking correctness properties and generating model instances using the model exploration tool Formula.

As advanced previously, the results presented in this paper are based on the work published by the authors of this paper in [1]. In this paper we provide an extended version of that work, presenting the development of our *CD2Formula* plug–in as additional contribution.

The remainder of the paper proceeds as follows. In Section II we provide a brief introduction to Formula. An overview of our framework is presented in Section III. Section IV presents the translation of a class diagram to Formula, while Section V describes the chosen OCL fragment and its representation into Formula. The automatic MDA–based translation of a UML class diagram to Formula, together with the development of our *CD2Formula* plug–in, is presented in Section VI. Section VII summarizes the strengths and weaknesses of our approach and discusses related work. Finally, Section VIII presents our main conclusions and future work.

II. A Brief Overview of Formula

In this section, we provide a general background of the Formula language by presenting the basic Formula concepts. In order to illustrate these basic concepts, we will lean on Figure 1, which, as we will explain later in detail, corresponds to an excerpt of a specific Formula domain we propose to define for translating class diagrams to Formula, but that we use here for explanatory purposes.

A. Formula units and design–space exploration

Formula distinguishes three units for modeling the problem: *domains*, *models* and *partial models*. Modeling in Formula always starts with specifying the *problem domain* and formalizing an abstraction of the problem that can be used by Formula to reason about the design [13]. A Formula *domain FD* is the basic specification unit in Formula for an abstraction and allows specifying ADTs and a logic program describing properties of the abstraction. The logic programming paradigm provides a formal and declarative approach for specifying such abstractions [9], which in Formula are represented by *rules* and *queries*. A Formula *model FM* is a finite set of data type instances built from constructors of the associated domain *FD*, and which satisfies all its constraints [9]. Formula allows to specify individual concrete instances of the design-space or parts thereof, in a specific Formula unit called *partial model* [9]. A Formula *partial model FPM* is a set of instance-specific facts placed along with some explicitly mentioned unknowns, which correspond to the parts of the model *FM* that must be solved. *FPMs* allow unknowns to be combined with parts of the model that are already fixed [9].

Finally, in order to explore the design–space, Formula loads the specification of the domains and the partial models defined for the specific problem and executes the logic program. The execution finds all intermediate facts that can be derived from the given facts in the partial model, and tries to find valid assignments for the unknowns. This step is carried out by the *Formula solver*, which, in case it finds a solution that satisfies all encoded constraints, will reconstruct a complete instance model from this information made of known facts [10], [11].

```
1 domain MetaLevel extends UserDataTypes {
2 Star ::= {star}.
3 primitive Class ::= (name: String, isAbstract: Boolean).
4 primitive Association ::= (name:String,
        srcType:Class, srcLower:Natural, srcUpper:UpperBound,
        dstType:Class, dstLower:Natural, dstUpper:UpperBound).
5 Classifier ::= Class + Association.
6 errorBadMultInterval := Association(_,_,srcLower,srcUpper,_,_,_),
                                          srcLower >srcUpper.
7 errorMetaDupAssoc := a1 is Association(name1,_,_,_,_,_,_),
                       a2 is Association( name2,_,_,_,_,_,_),
                       name1 = name2, a1 != a2.
8 ...
9 conforms := !errorBadMultInterval & !errorMetaDupAssoc & ...}.
```

Figure. 1: An extract of a Formula domain.

B. Domains' syntax

Basically, a Formula *domain* consists of *abstract data types*, *rules* and *queries*. Firstly, *abstract data types* constitute the key syntactic elements of Formula. Based on the defined data types, a number of *rules* and *queries* are specified as logic program expressions, ensuring the remaining constraints [9]. Roughly speaking, *rules* specify implications and *queries* restrict the valid states by specifying forbidden states.

Abstract data types. They are defined by using the operator *::=*, indicating in the right hand side their properties by means of *fields*. A data type definition can be labeled with the `primitive` keyword, denoting that it can be used for the extension of other type definitions. Otherwise, the data type results in a *derived constructor*. As a way of example, in line 3 of Figure 1 we define the `Class` data type representing the UML *Class* meta–element constructor. The derived type `Classifier`, on the other hand, is defined as the union of the `Class` and `Association` types (see line 5 of Figure 1).

Around data types, Formula defines different categorizations of the structural elements as building blocks for defining Formula expressions. These elements are mainly Formula *terms* and *predicates*.

As it can be inferred from the Help Formula Documentation [13], Formula distinguishes different types of *terms*, which could be established to be classified into two generalization groups: (1) simple and composite terms, and (2) what they call simply *Terms* (see Figure 2). On the one hand, Formula defines *simple terms* and *compound terms*. A *simple term* is represented by means of a type identifier containing variables, constants, or other simple terms as arguments, within parenthesis. As a way of example, in line 7 of Figure 1 we show the *simple term* `Association(name1,_,_,_,_,_,_)`, which represents all instances of the `Association` term, where the first parameter is set to the `name1` property. The other fields of this type (e.g., the `srcType`, `srcLower`, `srcUpper`, `dstType`, `dstLower` and `dstUpper` fields) are filled with a do not-care symbol ('_'), so that Formula will find valid assignments. A *compound term*, on the other hand, is represented by means of a type identifier with a list of *Terms* within parenthesis. As for the other generalization group, on the other hand, the building blocks of *Terms* are *atoms* (for

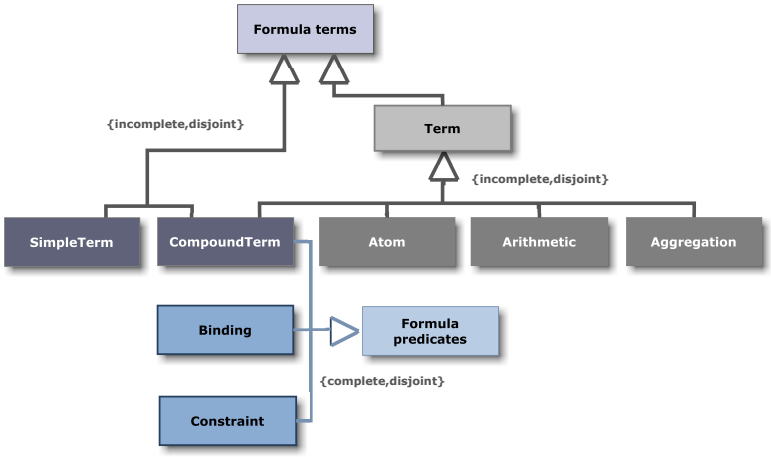


Figure. 2: Several of the Formula structural elements.

example the identifiers of variables and queries, explained later), *arithmetic* or *aggregation* expressions among other terms (`sum`, `count`, `max`, `min`, etc.), or *compound terms*.

All these different types of *terms* are, directly or indirectly, the basis for constructing *predicates*, which constitute the basic units of data, used for defining *queries* and *rules*. Among the different kind of *predicates*, we can note (see Figure 2): (1) *compound terms*, (2) *binding terms* (that is, gluing a variable to either a *type expression* or a *compound term*) and (3) *constraints*, which are defined by applying relational operators <,>=, !=, etc. among *terms*. An example of a binding term can be seen in line 7 of Figure 1, `a1 is Association(name1,_,_,_,_,_)`, where the variable `a1` is bound to the type expression `Association`. A constraint between terms is also shown in line 7, particularly in the expression `a1 != a2`.

Rules. They are specified by the operator `:-`, indicating, in the left hand, a simple term and, in the right hand, the list of *predicates* specifying the rule. A *rule* behaves like a universally quantified implication; whenever the relations on the right hand hold for some substitution of the variables, then the left hand holds for that same substitution [10], [11]. The intuition of rules is *production*; they create new entries in the fact-base of Formula, populating previous defined types with facts representing the members in the collection given in the rule.

Queries. Formula reserves a new syntax element for rules where left–hand side is a nullary construction [10], [11]. A *query* behaves like a propositional variable that is true if and only if the right hand side of the definition is true for some substitution [10], [11]. Queries are constructed by the operator `:=`, and can be also used like propositional variables to construct other queries. In particular, Formula defines in every domain the `conforms` standard query, where all constraints come together, and which defines how a valid instance of the domain have to look like. Based on the *existential quantification* semantics of queries, the *universal quantification* can be achieved by verifying the negation of a query representing the opposite of the original predicate. For example, in order to ensure that upper bounds of associations' multiplicities are upper than or equal to lower bounds, we firstly need to define a query representing the existence of associations verifying the opposite (see the definition of the query `errorBadMultInterval` in line 6 of Figure 1). With this query, we are considering such incoherent situation as a valid state. Thus, in order to verify that such situation is invalid, we include the negation ('!') of the query in the `conforms` query (line 9).

III. Encoding UML/OCL Models into Formula

As we have advanced previously, our proposal follows a MOF-like metamodeling approach, which is based on the framework the developers of the Formula tool give in [11]. In particular, the framework provided in [11] gives a representation in Formula of part of the key concepts defined both at the MOF meta–level [3], representing the M2 level, and at the instance–level [3], representing the M1 level for the object diagram. The resulted Formula expressions are grouped in an only Formula *domain*, which is used by the *Formula solver* to find, if it exists, a valid set of instances of arbitrary class diagrams at the M1 level (conforming with their MOF meta–level representation) and its corresponding instances at the M0 level (conforming with their instance–level representation). We remark that the authors in [10], [11] do not give a specific approach for the translation of OCL constraints.

Based on this proposal, we have extended and modified it giving weight to four main aspects. Firstly, we have mainly focused on obtaining a more faithful representation of the MOF structural distribution, specifying a richer metamodeling framework. Our extended proposal is materialized into four different Formula units distributed along the MOF meta levels, which ease the application and the understandability of our approach, while promoting units reutilization. Secondly, we provide an approach based on the CLP paradigm for analyzing model instances of specific class diagrams, and not arbitrary ones as authors in [10], [11] do, which we consider not enough when needed to reason about specific class diagrams. Thirdly, in contrast to [10], [11], we give an approach for translating OCL constraints to Formula by; (1) identifying a significantly expressive fragment of OCL, and (2) providing its translation into Formula. Finally, we have implemented part

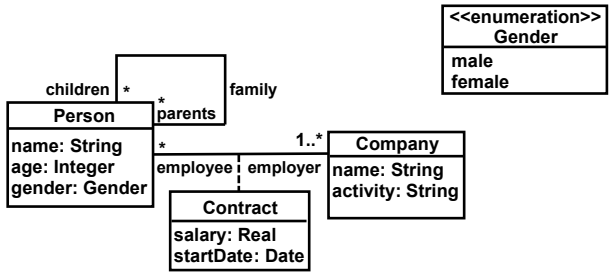


Figure. 3: Case study.

of our translation approach based on MDA, by means of the development of our *CD2Formula* Eclipse plug–in.

Each Formula unit defined in our approach contains two blocks of Formula expressions, related to the translation of the UML class diagram structural aspects (see Section IV) and its OCL constraints (see Section V), respectively. Our approach is illustrated with the case study of Figure 3, designed for explanation purposes covering basic aspects. In particular, this model describes both the contractual relationship between a "Company" and a "Person", and the family recursive relationship connecting the class "Person".

IV. Translation of a Class Diagram Structural Elements

In this section, we present a brief introduction of the rules we have defined to transform a class diagram ($\mathcal{CD}$), conforming with the UML metamodel [3] ($\mathcal{M}$), into Formula. Due to space reasons, in this paper we mainly focus on a set of basic structural UML class diagram features (UML *class*, *attribute*, *association*) for being frequently used for modeling structural aspects of systems, and also provide the translation of the UML *Classifier* element and *Association classes*. Next, we briefly explain their translation classifying the generated Formula instructions into the different MOF levels. For the explanation, we lean on Table I.

A. Classes, Associations and Properties

The translation of UML *Classes*, *Associations* and *Properties* into Formula follows the following proposal.

Level M2. For each meta model element *Class*, *Association* or *Property* ϵ $\mathcal{M}$, we define a primitive Formula data type with the same name and with specific *fields* (see level M2 in Table I). For example, in the case of classes, we define the data type `Class(;)` ϵ *CPS*, with two `String` fields (`name` and `isAbstract`). The definition of these data types allows Formula to create Formula instances representing specific UML classes, associations and types of properties, respectively, at the M1 level. In the case of the *Property* element ϵ $\mathcal{M}$, we define a type for each *build-in type*, called *typeName*`Property`, with specific fields (see Table I). In addition to `Integer`, `String` and `Boolean`, included in [11], we also give support to `Real`, `LiteralNull` and `UnlimitedNatural` types. The data type `HasProperty(;)` ϵ *CPS* is also defined to represent the possession of a property by a classifier.

Level M1. Two groups of expressions are defined at this level. **[M1a.]** Each specific *class*, *association* and *property* ϵ $\mathcal{CD}$, is represented by a Formula instance of the corresponding constructor (`Class`, `Association` or `Property` ϵ *CPS* defined at level M2). By these Formula instances, we are explicitly representing, in contrast to [10], [11], not arbitrary classes in a class diagram but specific ones. For example, the elements `ClassPerson` and `family` defined in `M1a` of Table I correspond to two Formula instances of the constructor `Class` and `Association`, respectively, defined at M2. In particular, specific properties ϵ $\mathcal{CD}$ are represented by a Formula instance of the corresponding `Property` constructor (e.g., *namePersonP is Str*`Property(...)` in `M1a` of Table I), and by an instance of the data type `HasProperty` ϵ *CPS*, representing the property's ownership (see Table I).

[M1b.] In order that Formula is able to generate instances of specific *class*, *association* and *property* ϵ $\mathcal{CD}$ to explore the concrete design–space, we need to create specific Formula data types representing each type of instance. For their definition, we have based on the description of the *Instances* package [3], in particular, on the *InstanceSpecification* element, for classes and associations, and on the *Slot* element, for properties. On the one hand, the definition of the UML *InstanceSpecification* element includes the classifier of the represented instance and the associated *InstanceValue* [3]. Taking this into account, for each *class c* ϵ $\mathcal{CD}$, we define a primitive Formula data type called `Instance`*c.name*`(;)` ϵ *CPS*, with two fields, representing the associated classifier and instance value, respectively (see level M1b in Table I). As a way of example, see the primitive data type `InstancePerson` in Table I. When the classifier is an association, the UML instance specification describes a *link* [3], so in this situations we name the created data types with the `Link` prefix. Since links connect class instances [3], for each *association a* ϵ $\mathcal{CD}$, we define a primitive Formula data type called `Link`*a.name*`(;;;)` ϵ *CPS*, which includes, additionally, the instance specifications of the associated classes (see for example `LinkFamily` in Table I). So that Formula can generate property's specific values, we define specific data types representing the property's slots, based on the specifications of the *Slot* element [3]. Taking this into account, for each *property* ϵ $\mathcal{CD}$, we define a primitive type called *p.name+p.owner.name*`Slot(;;)` ϵ *CPS* (e.g., `namePersonSlot` in Table I), which registers the owner, the property type and its value.

Level M0. Finally, in order that Formula can reason and search for valid instances of the specific classes, associations and properties of the source class diagram, we include the `Introduce(f,n)` command (used to add `n` terms of the element type `f`) with the corresponding `Instance`*c.name*, `Link`*a.name* or *p.name+p.owner.name*`Slot` data type, as `f`, and a specific number as `n`, to indicate the number of valid instances of such data type we want Formula to generate as part of the resulted object class diagram. For example, we define the `[Introduce(InstancePerson,2)]` command, so that Formula searches two valid instances of `InstancePerson` (see level M0 in Table I).

B. Classifier and Association Classes

A special remark have to be made regarding the *Classifier* element ϵ $\mathcal{CD}$ at the M2 level, and *association classes* ϵ $\mathcal{CD}$. On the one hand, the *Classifier* element is defined by means of a derived data type ϵ $\amalg \subset$ *CPS*, as the union of the `Class` and

Table. I: Excerpt of the CD to Formula mapping.

Level		Class	Association	Property
M2		**primitive Class ::= (name: String, isAbstract: Boolean).**	**primitive Association ::= (name: String, srcType: Class, srcLower: Natural, srcUpper: UpperBound, dstType: Class, dstLower: Natural, dstUpper: UpperBound).**	**primitive StrProperty::=(name:String, def:String, lower:Natural, upper:UpperBound).** ... **primitive LiteralNullProperty::=(name: String, def: Null,...). primitive UnlimitedNaturalProperty::=(name:String, def: UpperBound,.) Property::= StrProperty + ...+ userDataTypeProperties. primitive HasProperty ::= (owner: Classifier, prop: Property).**
M1	a	*Translation pattern:* **Class***c.name* **is Class(***"c.name", c.isAbstract***)** *Example:* **ClassPerson is Class("Person", false)**	*Translation pattern:* *a.name* **is Association(***"a.name",* **Class("***a.memberEnd.at(1).type.name***",** *a.memberEnd.at(1).type.isAbstract a.memberEnd.at(1).lowerValue, a.memberEnd.at(1).upperValue,* **Class("***a.memberEnd.at(2).type.name***",** *a.memberEnd.at(2).type.isAbstract a.memberEnd.at(2).lowerValue, a.memberEnd.at(2).upperValue)* Example: **family is Association("family",Class("Person",false), 0, 2, Class("Person",false), 0, star)**	*Translation pattern:* *p.name+p.owner.name***P is** *p.type***Property(***"p.name",p.default, p.lowerValue,p.upperValue)* **HasProperty(Class("***p.owner,..name***",** *p.owner.isAbstract),* *p.type***Property(***"p.name",p.default, p.lowerValue,p.upperValue)***)* *Example:* **namePersonP is StrProperty("name","",1,1) HasProperty(Class("Person",false),StrProperty("name","",1,1))**
	b	*Translation pattern:* **primitive Instance***c.name* **::= (id: Integer, type: Class).** *Example:* **primitive InstancePerson::=(id: Integer, type: Class).**	*Translation pattern:* **primitive Link***a.name* **::=(id: Integer, type: Association,** a.memberEnd.at(1).name: **Instance**a.memberEnd.at(1).type.name, a.memberEnd.at(2).name: **Instance**a.memberEnd.at(2).type.name). Example: **primitive LinkFamily::=(id:Integer,type:Association, child:InstancePerson, parent:InstancePerson).**	*Translation pattern:* **primitive** *p.name+p.owner.name***Slot ::= (owner:Element, prop:***p.type***Property, value***: valueType***)** *Example:* **primitive namePersonSlot::= (owner: Element, prop: StrProperty, value:String).**
M0		*Formula instructions pattern:* **[Introduce(Instance***c.name, number* **)]** *Example:* **[Introduce(InstancePerson,2)]** *Example of the Formula generated instances:* **InstancePerson(93,Class("Person",false)) InstancePerson(96,Class("Person",false))**	*Formula instructions pattern:* **[Introduce(Link**a.name, *number* **)]** *Example:* **[Introduce(LinkFamily,2)]** *Example of the Formula generated instances:* **LinkFamily(5, Association("family",Class("Person",false),0,2, Class("Person",false),0,star), InstancePerson(93, Class("Person",false)), InstancePerson(96, Class("Person",false)))**	*Formula instructions pattern:* **[Introduce(***p.name+p.owner.name***Slot***, number* **)]** *Example:* **[Introduce(namePersonSlot,2)]** *Example of the Formula generated instances:* **namePersonSlot(InstancePerson(93,Class("Person",false)), StrProperty("name","",1,1),202) namePersonSlot(InstancePerson(96,Class("Person",false)), StrProperty("name","",1,1),201)**

`Association` primitive data types so that we can generally refer to classes and associations. On the other hand, *association classes* ϵ $\mathcal{CD}$ are translated in the same way than associations but with the particularity of that they can have associated properties. In particular, since they are translate as associations, they can register the associated classes. Additionally, in our proposal we have defined slots in such a way that their owners are of type `Element` (see *M1b* translation of properties). This `Element` data type is defined as the union of `Instance` and `Link`, in such a way that we allow not only classes to have properties but also association classes.

Finally, the Formula expressions resulted from the translation of a $\mathcal{CD}$ are distributed into four different Formula units. On the one hand, Formula expressions defined at the *meta–model level* (M2) are included into a Formula domain called $MetaLevel_{FD}$. Since the representation of the meta–level M2 is the same whatever $\mathcal{CD}$ is considered, this Formula domain is defined once and used for each $\mathcal{CD}$. An excerpt of the $MetaLevel_{FD}$ domain has been presented in Figure 1. On the other hand, Formula expressions defined at the *model level* (M1) are distributed into two different units; the $CDModel_{FM}$ model, which is constituted by the Formula expressions defined in `M1a`, conforming with the $MetaLevel_{FD}$ domain, and the $InstanceLevel_{FD}$ domain, constituted by the expressions defined in `M1b`. Finally, the Formula expressions at the *instance level* (M0) are included in the $CDInstance_{FPM}$ partial model. Starting from these units, Formula can reason about the valid object class diagram, represented as instances of the elements of the $InstanceLevel_{FD}$ domain, conforming the given $\mathcal{CD}$, represented by means of the $CDModel_{FM}$ model.

V. Translation of Class Diagram Constraints

OCL integrity constraints undecidability has been tackled in the literature by defining methods that allow UML/OCL reasoning at some level. Examples of such methods are [6], [14]; (1) those that allow only specific kinds of constraints, (2) those that consider restricted models, (3) methods that do not support automatic reasoning, or (4) those that ensure only semi–decidable models. Our approach, which would fit within the first type, identifies a significantly expressive fragment of OCL and provides its translation to Formula for OCL constraints' decidable reasoning. In this section, we show that our OCL fragment can be formally encoded in Formula, thus, we guarantee finite reasoning for every OCL CD's constraint expressed using the constructors of our OCL fragment.

Our OCL to Formula translation relies on two foundations. Firstly, an OCL expression can be represented by means of First-Order Logic (FOL) expressions, taking into account that FOL, although less expressive than OCL, is commonly used for reasoning about the world using rules of deduction (see for example [15]). Secondly, a FOL expression can be translated into a logic constraint program *P*. More specifically, as stated in [16], each constraint logic program *P* can be translated in polynomial–time into first-order logic (FOL) according to its *Clark Completion* (from now on, we refer to the result of this translation as P^*). Roughly speaking, the *Clark Completion* of a program *P* corresponds to the completion of every atom or predicate symbol *p* in *P*. The *Clark Completion* captures the reasonable assumption that the rules for each atom or predicate symbol cover all of the cases where the atom is `true`. Taking this into account, the *Clark Completion* of an atom or predicate symbol can be represented as a combination of term expressions and rules, evaluated in variables, giving a `true` result. The inverse translation, that is, from the FOL representation of *P* (P^*) to *P* can be carried out by applying inverse versions of the *Clark Completion* algorithm, which compile specifications into the logic program it directly specifies, such as the one given in [17].

Based on these foundations, our proposal for the translation of OCL constraints is presented in Figure 4. Firstly, the OCL expression is translated to an equivalent FOL formula

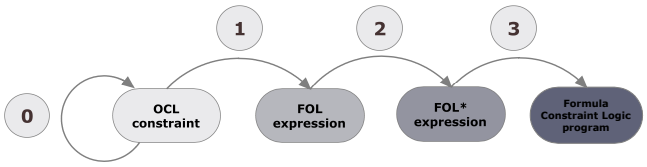


Figure. 4: Our approach for translating OCL constraints

(see step label 1 in Figure 4). Secondly, the predicates and expressions of the resulted FOL formula are rewritten in terms of Formula elements used to represent the class diagram as stated in our approach described in Section IV, obtaining the *Clark Completion* version of the final Formula logic program corresponding to the OCL constraint, from now on FOL* (see step label 2 in Figure 4). Thirdly, supported by the inverse algorithm of *Clark Completion*, we obtain the Formula logic program (see step label 3 in Figure 4). Finally, we can use the Formula automated tool for reasoning about the Formula model with the resulted expressions. Additionally, we note that it is not necessary to have the OCL constraint initially represented using the elements included in our OCL fragment, but, when possible, we can carry out a preliminary step where such OCL expression is redefined applying OCL equivalence rules (see step label 0 in Figure 4), resulted in other OCL expression whose constructs fit within our OCL fragment.

Next, we introduce the chosen OCL fragment and go on to explain our approach for translating it. More specifically, in order that the reader can get a better idea of such translating approach, firstly we will explain the translation of a simple OCL constraint, to serve as a reference explanation for the translation of the remainder elements of our OCL fragment.

A. Introduction to the Chosen OCL Fragment

Each OCL constraint is defined in the 'context' of a specific instance of the corresponding UML element, reserving the 'self' word to refer to the instance of the classifier on which the expression is evaluated. Taking this into account, an OCL invariant *I* has the form: `context C inv: expr(self)`, where `C` is the class $\epsilon\ \mathcal{CD}$ to which the invariant is applied and `expr(self)` is an OCL expression resulting in a `Boolean` value for each `self` ϵ `C`. In particular, this invariant states that, for *every* instance `self` of `C` existing in a system state, the property described by `expr` holds for `self`.

An OCL expression can be defined as a combination of *navigation paths* with OCL operations, which specify restrictions on those paths. A *navigation path* can be defined as a sequence of roles' names in associations (such as `p.children`, being `p` a `Person` instance in Figure 3), attributes' names (such as `c.name`, being `c` a `Company` instance in Figure 3), or operations (for example, `c.hireEmployee(p)`). Taking this into account, in Figure 5 we represent the syntax of our specific fragment, where `OCLExpr` is defined in a recursive manner. For example, an `OCLExpr` can be the result of applying relational operations to `AddExpr` expressions. Additionally, an `OCLExpr` can be the result of applying a boolean operation `BoolOper` to a `Path`, or a `Path` to which a `SelectExpr` is applied. An `OCLExpr` can be also constituted by boolean combinations of these OCL expressions (`not`, `and` and `or`). A `Path` expression represents the structural way of defining navigation paths, starting from a

```
OCLExpr    ≡RelExpr |Path BoolOper |Path SelectExpr
            not OCLExpr | OCLExpr1 and OCLExpr2
            OCLExpr1 or OCLExpr2
Path       ≡PathItem | PathItem.Path
PathItem   ≡role | classAttr | operation
            roleName.role | roleName.classAttr
            roleName.oper | roleName.transClosuOper
RelExpr    ≡AddExpr <,<=,>,>=,=,!= AddExpr
AddExpr    ≡MulExpr | AddExpr +/- AddExpr
MulExpr    ≡Path | MulExpr * Path | MulExpr/Path
SelectExpr ≡-> select( OCLExpr) BoolOp|
            -> select( OCLExpr) SelectExpr
BoolOper   ≡  -> size()| ->  forAll(OCLExpr)
```

Figure. 5: Syntax of the OCL fragment.

`PathItem`, by combining roles' names, attributes' names or operations, with the dot operator. For an explanation of OCL, we refer to [4].

B. Our Translation Approach

Formula does not have a concept similar to that of OCL invariants but gives the possibility of defining queries, which provide a way to represent invariant semantics. As way of example of our approach, in this section we introduce the basic rule for translating OCL invariants where the `OCLExpr` corresponds to a simple relational expression `RelExpr`. We explain this rule by applying it to the invariant `context Person inv: self.age >=18`, which formalizes the constraint "The people registered in the system must be older than 18 years old" (see Table II).

First–step. This step is carried out by means of an interpretation function *FOL*(), which translates each OCL expression `expr(self)` defined in an instance `self` ϵ `C`, into a First–Order Logic (FOL) formula defined in the variable `self` (see label (1) in the first step of Table II). Basis in first order logic states that the universal quantifier corresponds to a negated existential, so the previous expression is equivalent to the one label (1'), where *FOL*`(not expr(self))`, corresponds to the mapping of `not expr(self)` into First–Order Logic.

Second–step. As described previously, each constraint logic program *P* can be translated into first-order logic (FOL) according to its *Clark Completion P** [16]. Roughly speaking, the *Clark Completion* of an atom or predicate symbol can be represented as a combination of term expressions and rules, evaluated in variables, giving a `true` result.

Taking this into account, the second step is devoted to represent the semantics given by the affirmative evaluation of *FOL*`(not expr(self))` in the collection of instances `self` $\in$ `C`, by means of Formula expressions. Since paths in OCL are defined in terms of instances of the class diagram, and in our approach such instances are defined by means of the data types defined in the $CDInstance_{FPM}$ partial model, such Formula expressions are written in terms of the `Instance`*className*, `Link`*associationName* and/or *propertyName*+*ownerName*`Slot` data types. Based on this premise, in this second step we rewrite the FOL expression *FOL*`(not expr(self))` in terms of Formula expressions by applying a second function called *FOL**(). This function *FOL**() basically represents the predicate *FOL*`(not expr(self))` by using the corresponding Formula terms and predicate symbols $\in$ $InstanceLevel_{FD}$, and Formula constraints, in such a way that

Table. II: Translation of an invariant and example of use.

Translation of a `RelExpr` invariant	
OCL Invariant: `context C inv: expr(self)`	
First–step:	$\forall$`self` $\in$ `C` *FOL*`(expr(self)).` (1) $\neg(\exists$`self` $\in$ `C` *FOL*`(not expr(self)).` (1')
Second–step:	$\neg$(*FOL**`(C)` *FOL**`[`*FOL*`(not expr(self))]`) (2)
Third–step:	`query:=`*CLP*(*FOL**`[`*FOL*`(not expr(self))]`) `conforms := ! query.` (3)
Example of application	
OCL Invariant: `context Person inv: self.age >=18`	
First–step:	$\forall$`self` $\in$ `Person age(self)>=18.` (1) $\neg(\exists$`self` $\in$ `Person age(self)<18).` (1')
Second–step:	$\neg(\exists$`self` $\in$ `InstancePerson(id,type)` `agePersonSlot(self,def,val)` `val<18)`.(2)

Third–step:	`query:=agePersonSlot(self,_,val),`
	`val<18.`
	`conforms := ! query.` (3)

Table. III: Translation of part of our OCL fragment.

OCL expression	Translation approach
`E1 and E2`	*CLP*(*FOL**(*FOL*`(E1)`))&*CLP*(*FOL**(*FOL*`(E2)`))
`E1 or E2`	*CLP*(*FOL**(*FOL*`(E1)`))|*CLP*(*FOL**(*FOL*`(E2)`))
`not E`	*CLP*(*FOL**(*FOL*`(not E)`))
`C-> size()`	`count(`*CLP*(*FOL**(*FOL*`(C)`))`.`
`C-> forAll(c|exp(c)`	`query:=`*CLP*(*FOL**(*FOL*`(not exp(c))`)). `conforms:= ! query.`
`C-> select(c|exp(c)`	$S_{C,expr}$`Type::=(self:`T_{self}`,sele:`T_{sele}) $S_{C,expr}$`Type(self,sele):-` *CLP*(*FOL**(*FOL*`(exp(c))`))

the resulted expression is evaluated to `true` (see step labeled (2) in Table II). In particular, the application of this step to our constraint consists of representing `age(self)<18` in terms of the `agePersonSlot` whose `val` property is less than 18.

Third–step. Taking into account the semantics of queries in Formula, the FOL expression given in the second step is finally represented by means of the definition of a `query` and the verification of its negation in the `conforms` query (see step labeled (3) in Table II). This step is materialized by means of the application of the function *CLP*(), which basically rewrites the terms resulted from (2), and joins them by ','.

To sum up, the translation of an invariant I is carried out by means of the composition of the three functions, *CLP* $\circ$ *FOL** $\circ$ *FOL*`()`.

C. Translation Approach of More Complex OCL Expressions

Having presented our approach for the translation of a simple OCL invariant, next we make some remarks regarding the translation of the remainder elements in our OCL fragment. More specifically, in Table III we present the translation rules we define for the *conjunction*, *disjunction* and the *negation* operators considered in the OCL fragment. For example, we describe the translation of an OCL expression with the *conjunction* operator `E1 and E2` as: *CLP*(*FOL**(*FOL*`(E1)`)) & *CLP*(*FOL**(*FOL*`(E2)`)), where each expression is translated recursively using the translation rules presented in the rest of this paper by applying the defined functions. In particular, if *CLP*(*FOL**(*FOL*`(E1)`)) results in the verification of a query `!query1` in the `conforms` one, and *CLP*(*FOL**(*FOL*`(E2)`)) results in the verification of another query `!query2`, the result of translating the conjunction is the expression `!query1 & !query2` specified in the `conforms` query (that is, `conforms:= !query1 & !query2`). The translation of the *disjunction* operator, on the other hand, results in the expression `conforms:= !query1 | !query2`, being `!query1` and `!query2` the translations of `E1` and *CLP*(*FOL**(*FOL*`(E2)`)), respectively. Finally, the translation of the *negation* operator results in the expression `conforms:= !query`, being `!query` the translation of `not E1`.

The remainder OCL expressions in our framework include operations in collections. Excluding the `select` and `transitive closure` elements, whose translation requires extra attention, we consider that the translation of the remainder OCL elements (`forAll` [4](p. 29, Section 7.7.3) and `size` [4] (p. 157, Sec. 11.7.1)) can be easily understood by considering our previous explanations. Next, we briefly describe our approach for their translation into Formula.

Select operation. Since this operation refers to obtaining a subcollection from a set of elements ([4] pp. 27, Sec. 7.1.1), its translation consists of defining a new Formula data type and populate it with the facts representing the members in the collection we want to select (see the first and second lines, respectively, of the translation of the `select` operation in Table III). As a way of example, if we want to collect the female employees of a company, we define the type: `FemaleEmp::= (self: InstanceCompany, sele:InstanceEmployee)`, and populate it by means of the following rule, which gathers only female employees:

```
FemaleEmp(self,sele) :-
          LinkContract(_,_,sele,self),
          genderPersonSlot(sele,_, val),
          val=female.
```

Transitive closure. Transitive closure is normally needed to represent model properties which are defined in a recursively fashion. The translation of closures is not straightforward since they are not finitely axiomatizable in first order logic, and OCL also does not support them natively [18]. Nevertheless, it is possible to define the transitive closure of relations that are known to be finite and acyclic. In particular, for its translation we have based on both, the definition of transitive closure provided in [18], and the representation in CLP of acyclicity constraints provided in [11] (p. 3), and proposed a translation based on defining Formula rules, considering the fact that CLP exposes fixpoint operators via recursive rules. Additionally, the translation of this operation allows us to support *aggregation*.

Finally, the Formula model resulted from the translation of a class diagram model annotated with OCL constraints (that is, the 4 Formula units including the Formula translation of the OCL constraints), is used by Formula for reasoning about it. More specifically, the tool inspects the Formula model looking for a valid and non–empty instantiation of the CD/OCL model to proof its satisfiability. If the result is empty, the defined CD/OCL model is not satisfiable. Otherwise, Formula proposes a conforming instantiation model of the defined CD/OCL model, according to the desired software system settings.

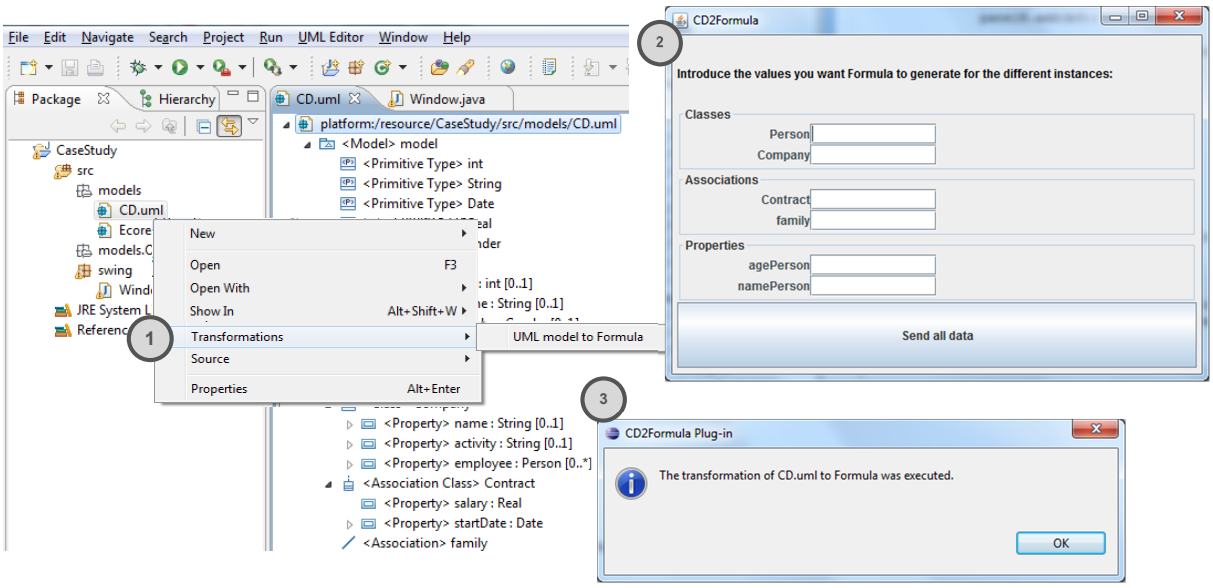


Figure. 6: A snapshot of the *CD2Formula* plug–in.

VI. Automatic Translation

In order to manually transform a class diagram into the Formula language, a professional with both UML and Formula skills may be required. Additionally, such an encoding process may entail a big effort depending on the class diagram used. The challenge is to perform such a transformation in a viable and cost–effective way. The complexity of some software designed models together with their possibility of change over time, make the manual transformation of every class diagram representing a software model into the input language of a model finder tool, a cumbersome and costly endeavor. To overcome these challenges, we have based on an MDA tool-approach to automatically carry out the translation of a class diagram to Formula. More specifically, we have developed an Eclipse plug–in, called *CD2Formula* plug–in, which gives support for the class diagram to Formula transformation as stated in our proposal (see in Figure 6 a snapshot of the plug–in). The idea is that the defined plugin together with the Formula tool, constitute the complete proposed framework for class diagram to Formula specification. Firstly, by means of the *CD2Formula* plugin we automatically generate, from a class diagram, the Formula specification, which is taken by the Formula model finder for reasoning about the input class diagram model.

The core of our plug–in is that it itself uses a MDA–based plug–in that gives support for customizable model–to–text (M2T) transformations. Among the large amount of MDA-based tools in the literature, we have chosen the MOFScript Eclipse plug-in, which we have already used in previous works [19], [20]. MOFScript is an Eclipse plug–in [21], [22] that implements the MOFScript language, which was one of the candidates in the OMG RFP process on MOF Model-to-Text Transformation [23]. As input models, MOFScript can use any model that complies with the EMF [24] metamodel. From these input models, the tool can generate any arbitrary text (such as Java code or XML) by using a defined set of MOFScript transformations. Each MOFScript transformation consists of transformation rules that are basically the same as functions, and which define the behavior of the transformation. The transformation rules are defined based on the metamodel and subsequently compiled and executed on the model.

In our particular case, we use the UML 2.0 metamodel and a class diagram that represents the software design as the model. To create class diagram models, we can use any UML 2.0 compliant tool that can create models, as `.uml` extension files, in the XMI format supported by EMF (e.g., the UML2 Eclipse plug-in [25]).

As far as the Formula program generation is concerned, an important remark must be made. In our proposal for the Formula representation of a UML class diagram, we need to include specific Formula instructions to tell the Formula solver the number of valid instances (for example, the number of class and association instances), we would like for the final solution. Such number of instances is set by means of the `Introduce` Formula instructions, which, in our particular case, are included in the $CDInstance_{FPM}$ partial model defined for each class diagram. Since such number of instances should be established by the user before carrying out the transformation from the class diagram to the Formula specification, we firstly need to ask the user for such information, which is specific for each class diagram. Taking this into account, we have defined two sets of MOFScript transformations, devoted respectively to: (1) generate a java GUI (Graphical User Interface), which ask the user for the required information, and (2) create the Formula specification for the class diagram (whose $CDInstance_{FPM}$ partial model is generated taking into account the values inserted by the user by means of the previously generated GUI interface). Both sets of MOFScript transformation files are devoted to produce the print statements that generate the java GUI interface and the different Formula units, respectively.

Particularly, in the definition of the MOFScript transformations, we have followed a concrete idea, which consists on defining two kinds of transformation rules: (1) those that tra-

```
module::CDToCDModel(){
   standardClassDiagram()
   var i:integer
   var upper:String
   var propertyType: String
   //Create an instance of the Class element for each class in the CD
   classes -> forEach(c:uml.Class){
     println('   class'+c.name+' is Class("'+c.name+'", '+c.isAbstract+')')
   }
   //Create an instance of the Association element for each association in the CD
   associations -> forEach(a:uml.Association){
     i=0
     println('   '+a.name+' is Association("'+a.name+'", ')
     print('      ')
     a.memberEnd -> forEach(p:uml.Property){
         i=i+1
         if(p.upper==-1)
            upper='star'
         else
            upper=p.upper
         print('class'+p.type.name+', '+p.lower+', '+ upper)
         if(i==1)
             print(', ')
      }
     println(')')
   }
   //Create an instance of the Property element for each property in the CD
   ...
   println('}')
}
```

Figure. 7: An extract of a Mofscript rule.

verse the model and collect the information in it and (2) those that generate actual code (java print statements or Formula structures, respectively). The first kind gathers data and records them in collections (such as lists or hashtables) or other built-in types. Finally, the code generation rules use this information in print statements. As a way of example, an extract of one of the defined rules is shown in Figure 7. In particular, this rule called `CDToCDModel` will create the Formula expressions that constitute the $CDModel_{FM}$ model (such as `classPerson is Class('Person', 'false')`).

Regarding the MOFScript transformation files defined to generate the Formula units, we have created 3 files: `main.m2t`, `helpers.m2t`, and `FormulaUnits.m2t`. The transformation file `main.m2t` contains the main rule that actually generates the complete Formula specification of the class diagram by using specific rules from the rest of the defined transformation files. The file `helpers.m2t` has been defined to be used as a library, containing commonly used rules that are required by other rules during the transformation process. Finally, the `FormulaUnits.m2t` file contains the rules devoted to finally produce the print Formula structures that constitute the three Formula units in our approach, which depend on the specific class diagram.

As for the generation of the GUI interface for a specific class diagram, we have defined an only MOFScript transformation file called `main.m2t`. This file defines MOFScript rules that mainly traverse the class diagram and generate the java print statements that define the different `labels` and `text fields` of the form, together with a button to send the inserted data (see the GUI interface `CD2Formula` labeled 2 in Figure 6) to be used for the second transformation. In particular, the second group of MOFScript files gets the data given by the user through the GUI interface, thanks to a specific MOFScript's functionality. MOFScript allows the possibility of invoking java methods from MOFScript rules and retrieving the returned information. Taking this functionality into account, the java GUI interface file contains a specific java method in such a way that, when the form in the java GUI interface is filled out, such method returns a hashtable with the pairs (*type of instance–number of instances desired*). In this way, the second group of MOFScript transformation files takes such information to print the corresponding `Introduce` Formula instructions of the final Formula specification.

We want to highlight that since the MOFScript transformation rules are defined based on our transformation rules between a class diagram and the Formula model, and these transformation rules are defined independently of the class diagram used, the MOFScript transformations do not have to be modified to translate a different class diagram.

Having defined the two sets of MOFScript transformation files, we have developed an Eclipse plug–in called *CD2Formula* in such a way that it integrates such MOFScript rules so that the transformation from a class diagram to its Formula specification can be generated in an automatic fashion. More specifically, the plug–in provides a menu option available for each UML class diagram (specified as `.uml` extension files), which allows the execution of the MOFScript transformations. The transformation process encompasses three steps. Firstly, the user chooses the menu option the plug–in provides, which executes the first set of MOFScript transformations that lead to the dynamic creation of the GUI interface. Secondly, the plug–in refreshes the Eclipse project so that the corresponding interface java class can be created and instantiated. Thirdly, the second set of MOFScript transformations is executed, which (1) leads to the invocation of the java method that shows the interface, asking the user for the required values, (2) retrieves the values inserted by the user in the interface, and (3) generates the Formula specification (that is, the `FormulaSpecifications.4ml` file), using such values. Finally, the resulted file is used by the Formula tool for reasoning about the class diagram.

About the usability of the proposal and the developed plug–in, we have to say that it is only required a professional with OCL skills in order to be able to apply our OCL to Formula translation proposal to the OCL constraints defined in the specific class diagram. Excluding it, no specific knowledge would be required for managing the plug-in since its interface has been developed so that it is simple and easy to use.

VII. Discussion and Related Work

As described previously, the formalization and analysis of UML class diagrams can be done by means of translating the model to other language that preserves its semantics, and finally, using the resulted translation to reason about the design. Taking into account that there is not an only language for materializing such translation, and that several translation approaches can be established using a same language, a discussion about the semantic support of the language, together with the strengths and weaknesses of the particular translation approach, is worthwhile. Our work bets on using Formula for the semantics preserving translation of the models to be verified. As for the use of Formula instead of other analyzers, in particular, Formula authors present in [11] a comparison with other tools, both SAT (Boolean Satisfiability) solvers and alternatives such as *ECLiPSE* and *UMLtoCSP*, focusing mainly on Alloy [26], for being the closest tool to Formula. Although the Formula authors provide a careful comparison with Alloy in [11], it is worth noting the strengths of Formula, such as a more expressive language or its model finding problems, which are in general undecidable.

Our approach follows a multilevel MOF-like framework based on the one proposed in [11]. On the one hand, we propose a more faithful representation of the basic UML metamodel and instance domain elements [3]. We consider that providing a translation that captures the structural distribution of the MOF architecture can contribute to ease the application and understandability of the representation of a CD/OCL model into Formula. We also give support for the translation of more metamodel elements (such as full support to generalization, property types other than *Integer*, *String* and *Boolean*, including user defined data types, property's multiplicities, etc.), thus providing a richer framework. Additionally, we enhance the proposal given in [11] by identifying an expressive fragment of OCL, which guarantees finite satisfiability and providing a formalization of the transformations from such OCL fragment to Formula. At this respect, several related works can be cited, being one of the most complete proposals the one given in [14]. In [14], the authors define a fragment of OCL called OCL–lite, and prove the encoding of such a fragment in the description logic $\mathcal{ALCI}$, so that Description Logic techniques and tools can be used to reason about class diagrams annotated with OCL–lite constraints. A difference of this approach with ours is the fact that, although the chosen fragment is quite similar than ours, we have tried to identify a simplest fragment so that no element included in it can be inferred from other constructors in the fragment by applying direct OCL equivalences (such as the `implies` operator). In our particular case, there are several OCL operations and expressions whose representation in Formula is straightforward by applying equivalences (such as the `exists` [4] (p. 30, Sec. 7.7.4), `isEmpty/notEmpty` [4] (p. 157, Sec. 11.7.1), `xor` [4] (p. 153, Sec. 11.5.4), or `reject` [4] (p. 27, Sec. 7.7.1)).

On the other hand, there are other elements, such as `oclIsTypeOf`, which is considered in the OCL–lite fragment but that can not be represented into formula. More specifically, Formula does not support the translation of, for example, the following OCL properties [4] (p. 146, Sec. 11.3); (1) `oclIsTypeOf(t : OclType)`, which is used to know whether the object to which it is applied is of type `t`, and (2) `oclIsKindOf(t : OclType)`, which returns `true` whether `t` is either the direct type of the object to which the operation is applied or a supertype of the object. As for the representation in Formula of the `oclIsTypeOf` operation, there is no way to know the type of a variable by using the Formula syntax, but the mismatch among variables and types is verified by the Formula checker. The same argument is applied to the `OclIsKindOf` operation. Similarly happens with OCL operations that are state dependent (such as the operations `oclIsInState(t: OclType)`, which evaluates whether the object is in a specific state, and `oclIsNew(t: OclType)`, which checks whether the object does not exist in the previous state of the system but exists in the current state). In both cases, a UML statemachine diagram is required, and although representing UML statemachines in Formula could constitute an interesting issue for future work in order to give support to reason also about dynamic system models, it is out of the scope of this work. Focusing on reasoning about static class diagrams models, in spite of these operators, we give support to other not straightforward operators, such as *transitive closure*, not normally included in related works.

VIII. Conclusion and Future Work

We present an overall framework to reason about UML/OCL models based on the CLP paradigm, using Formula. Our framework provides a way to translate a UML model into Formula, following a MOF-like approach. We also identify an expressive fragment of OCL, which guarantees finite satisfiability and we provide an approach for translating it to Formula. We also provide an implementation of our UML to Formula proposal by the development, following a Model Driven Architecture (MDA) approach, of the *CD2Formula* plug–in. Particularly, starting from a UML class diagram representing the static structure of a software system, our plug–in carries out the automatic generation of the Formula specification corresponding to such UML model, by simply choosing a menu option the plug–in provides. The proposed framework can be used to reason, validate and verify UML software designs by checking correctness properties and generating model instances using the model exploration tool Formula.

Although we support the automatic translation from a UML class diagram to Formula by means of our plug–in, the automatic translation to Formula of specific class diagram's OCL constraints specified using our OCL fragment constitutes a remaining work.

Acknowledgments

This work has been partially supported by the Academy of Finland, the Spanish Ministry of Science and Innovation (project TIN2009-13584), and the University of La Rioja (project PROFAI13/13).

REFERENCES

[1] B. Pérez and I. Porres, "Reasoning about UML/OCL models using constraint logic programming and MDA," Proceedings of the 8th International Conference on Software Engineering Advances (ICSEA'13), 2013, pp. 228–233.

[2] J. Bézivin,"Model driven engineering: an emerging technical space," Proceedings of the International Summer School of Generative and Transformational Techniques in Software Engineering (GTTSE'05), 2006, LNCS, vol. 4143, pp. 36–64.

[3] OMG, UML 2.4.1 Superstructure Specification, Document formal/2011-08-06. Available at: http://www.omg.org/. Last visited on May 2014.

[4] OMG, Object Constraint Language, Version 2.3.1, OMG Document Number: formal/2012-01-01. Available at: http://www.omg.org/spec/OCL/2.3.1/PDF. Last visited on May 2014.

[5] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini, "A formal framework for reasoning on UML class diagrams,", Proceedings of the 13th International Symposium of Foundations of Intelligent Systems (ISMIS'02), LNCS, vol. 2366, 2002, pp. 503-513.

[6] J. Cabot, R. Clarisó, and D. Riera, "Verification of UML/OCL class diagrams using constraint programming," Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08), IEEE Computer Society, 2008, pp. 73–80.

[7] V. Del Bianco, L. Lavazza, and M. Mauri, "Model checking UML specifications of real time software," Proceedings of the 8th International Conference on Engineering of Complex Computer Systems (ICECCS 2002), IEEE Computer Society, Los Alamitos, 2002, pp.203-.

[8] V. Del Bianco, L. Lavazza, and M. Mauri, "A formalization of UML statecharts for real-time software modeling," The 6th Biennial World Conference On Integrated Design Process Technology (IDPT 2002), "Towards a rigorous UML" session, Pasadena. 2002.

[9] Formula - Modeling Foundations, Website: http://research.microsoft.com/en-us/projects/formula. Last visited on May 2014.

[10] E. K. Jackson, T. Levendovszky, and D. Balasubramanian, "Reasoning about metamodeling with formal specifications and automatic proofs," Proceedings of the 14th International Conference of Model Driven Engineering Languages and Systems (MODELS 2011), 2011, pp. 653-667.

[11] E. K. Jackson, T. Levendovszky, and D. Balasubramanian, "Automatically reasoning about metamodeling," Software & Systems Modeling, February, 2013, doi:10.1007/s10270-013-0315-y.

[12] OMG, OMG Model Driven Architecture, Document omg/2003-06-01, 2003, Available at: http://www.omg.org/. Last visited on May 2014.

[13] Formula 1.3. Formula documentation (Help). Formula downloads, available at: http://research.microsoft.com/en–us/downloads/49f1072b–c0ec–4b1e–bdd7–4661ea07b5b3/default.aspx. Last visited on May 2014.

[14] A. Queralt, A. Artale, D. Calvanese, and E. Teniente, "OCL-Lite: A decidable (yet expressive) fragment of OCL*," Proceedings of the 25th International Workshop on Description Logics (DL'12), Description Logics, vol. 846, 2012, pp. 312-322.

[15] B. Beckert, U. Keller, and P. H. Schmitt,"Translating the object constraint language into first-order predicate logic," Proceedings of the Workshop at Federated Logic Conferences (FLoC02), 2002, available at i12www.ira.uka.de/ key/doc/2002/BeckertKellerSchmitt02.ps.gz.

[16] J. Jaffar, M. J. Maher, K. Marriott, and P. J. Stuckey, "The semantics of constraint logic programs," J. Log. Program., vol. 37, 1998, pp, 1-46.

[17] A. Bundy, "Tutorial notes: reasoning about logic programs," Proceedings of the 2nd International Logic Programming Summer School (LPSS'92), LNCS, vol. 636, 1992, pp. 252-277.

[18] T. Baar, "The definition of transitive closure with OCL - limitations and applications," Proceedings of the 5th Andrei Ershov International Conference in Perspectives of System Informatics (PSI'03), LNCS, vol. 2890, 2003, pp. 358-365.

[19] B. Pérez, "Towards decision facts management systems: the particular case of clinical guidelines," PhD thesis, Department of Computer Science and Systems Engineering, University of Zaragoza, Spain, 2011.

[20] B. Pérez and I. Porres, "Authoring and verification of clinical guidelines: a model driven approach", Journal of Biomedical Informatics, vol. 43, num. 4, 2010, pp. 520-536.

[21] J. Oldevik,"MOFScript eclipse plug-in: metamodel-based code generation," Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France, 2006.

[22] MOFScript user guide, version 0.6 (MOFScript v 1.1.11), 2006, Available at: http://www.modelbased.net/mofscript/docs/MOFScript-User-Guide.pdf. Last visited on May 2014.

[23] OMG, OMG document ad/2005-11-03. MOFScript second revised submission to the MOF model to text transformation RFP (2005), Available at: http://www.omg.org/. Last visited on May 2014.

[24] EMF development team, The eclipse modeling framework website: http://www.eclipse.org/modeling/emf/. Last visited on May 2014.

[25] The Eclipse UML2 project, Website: http://www.eclipse.org/modeling/mdt/?project=uml2. Last visited on May 2014.

[26] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "UML2Alloy: a challenging model transformation," Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS'07), LNCS, vol. 4735, 2007, pp. 436-450.

Simulation-Based Optimization for Software Dynamic Testing Processes

Mercedes Ruiz
Department of Computer Science and Engineering
University of Cádiz
Cádiz, SPAIN
e-mail: mercedes.ruiz@uca.es

Javier Tuya
Department of Computing
University of Oviedo
Gijón, SPAIN
e-mail: tuya@uniovi.es

Daniel Crespo
Department of Computer Science and Engineering
University of Cádiz
Cádiz, SPAIN
e-mail: dani.crespobernal@alum.uca.es

Abstract - **Managing software development projects requires the coordination of different processes that may be performed by different teams, e.g., a development team and a separate testing team. This coordination aims at optimizing the trade-off between cost, schedule and delivered quality. Simulation models are a powerful tool to explore what-if scenarios that help managers to achieve this trade-off and to fine-tune different project parameters. This paper presents a simulation model based on a multi-paradigm approach that connects development and testing processes. The testing process model is based on the process model described in the ISO/IEC/IEEE 29119-2:2013 standard. The simulation model is built using two different methods: the discrete-event approach, to simulate the execution of the dynamic testing processes, and the agent-based approach, to in-depth simulate defects life cycle. Results show how the simulation model is used to explore the evolution of a number of process metrics. Then, the simulation model is used to determine the resource distributions in order to optimize two relevant process metrics: the efficiency of the testing process and the average defect life.**

Keywords - software testing; multiparadigm simulation; test management; test process optimization.

I. INTRODUCTION

This is a revised and augmented version of our previous work, which appeared in the Proceedings of the Seventh International Conference on Software Engineering Advances (ICSEA 2012) [1]. Software testing is concerned with planning, preparation and evaluation of software products and related work products to: a) determine that they satisfy specified requirements, b) demonstrate that they are fit for purpose and c) detect defects [2]. In general, testing can be viewed as a means of improving the quality of a given product and mitigating risks due to poor quality.

Testing can be carried on using different approaches (e.g., scripted or exploratory), at different levels (e.g., unit, system, integration or acceptance), using different techniques and tools and with different degrees of independency (ranging from testing performed by the producer to third party testing). When testing entails the execution of the system under test, it is often referred to as dynamic testing. Testing exists in an organizational context and is carried on a given project or service. Therefore, the testing activities are tightly interrelated with the development ones, and both shall be planned, monitored and controlled. Problems of quality of the system under test or delays in the development hamper the testing process. Conversely, an inadequate or delayed testing endangers the development process. If not managed properly, both development and testing processes may jeopardize the goals of cost, schedule and quality of a project.

Both development and testing can be described as processes and take advantage of the use of simulation models for helping project and/or test managers in daily tasks of planning, monitoring and control.

Informally, a simulation model can be considered as an abstract view of a complex system comprised of a set of rules that tell how to obtain the next state of the system from the current state. Those rules can be of many different forms: differential equations, state charts, process flowcharts, schedules, etc. The outputs of the model are produced and observed as the model is running.

There is much research on simulation models of the software development process [3]. However, there is lesser research on simulation models for the testing process, usually at the unit level. Furthermore, when testing is considered as part of a simulation model of the development process, it is often over simplified. The goal of this paper is to devise a multi-paradigm simulation model for the testing process to gain insights in how the testing process influences the goals of a given project. The simulation model can be used to simulate the testing process at the system level and to help in decision-making in the test managing processes.

Contributions of this paper extend previous work [1] and include:

1. A multi-paradigm simulation model of the dynamic testing processes, which combines a discrete-event model and agent-based model.
2. The optimization of two key variables of the testing process: process efficiency and average defect life

The main contribution of this work is a multi-paradigm simulation model of the dynamic testing processes that

combines a discrete-event model and agent-based model. The model can be used to simulate the testing process at the system level and to help in decision-making in the test managing processes.

The structure of the paper is as follows: Section II shows the works related to our proposal; Section III introduces the multi-layer process model proposed by the International Standards group upon which our simulation model is based; Section IV describes the simulation model; Section V shows two simulation optimization scenarios. Finally, our conclusions and further work are given in Section VI.

II. RELATED WORK

During the recent years a lot of research has been done in the field of software testing. These studies are mainly oriented to enhance and optimize the software testing process improving the results obtained after the development of software projects. Several techniques and methods have been used to reach this goal. Knowledge Management has been a recurrent tool due to its usefulness for revising software testing processes [4], learning from the errors committed in the past [5], collecting, analyzing and managing lessons learned [6] and improving the quality of software testing [7].

Sometimes, it is interesting to study the behavior of processes in order to detect weaknesses so that improvement can be effectively performed. Process modeling techniques have been widely used to address these issues. Models can be used to estimate process outcomes such us the number of defects remaining and the time required to detect defects either for a subsequent optimization [8], to estimate effort, cost and schedule [9] or to perform cost control management [10]. Modeling also plays an important role in decision-making support. Reference [11] presents a goal-driven measurement model for software testing process so that software organizations can deduce the appropriate measurement process according to the process goals they determine. On the other hand, reference [12] provides a quantitative defect management model that can be improved to be practically useful for determining which activities need to be addressed to improve the degree of early and cost-effective software fault detection with assured confidence. Finally, reference [13] proposes a competence model that could be applied to train staff in software testing activities and to recruit the appropriate profiles improving their performance.

Some authors have developed their own frameworks to study software test processes. Reference [14] describes a conceptual framework to specify and explicitly evaluate test process quality aspects and [15] proposes a software testing improvement framework based on the Plan–Do–Check–Act (PDCA) method.

Some other techniques employed for software testing process improvement include Bayesian networks for process evaluation [16], Markov decision models to optimize software testing by minimizing the expected cost with given software parameters of concern [17], multi-objective feature prioritization for testing planning and controlling [18], system dynamics to formulate and quantify the software testing processes [19] and even the usage of software engineering standards to improve the testing process [20].

Although the above mentioned techniques are extremely useful to improve the software testing processes, sometimes it is necessary to look into the processes with more detail. In order to effectively optimize a process the current behavior must be examined. Furthermore, all the possible variations suffered by the process due to the different scenarios that may occur, should be taken into account in order to analyze the results derived from one situation or another. Software process simulation provides the means to accomplish this goal in a cost effective way.

The search string “simulation” AND “software testing process” AND “management” and others alike used in several digital libraries and citation databases of peer-reviewed literature retrieves only a few number of papers. In many of the papers retrieved, the term “simulation” is frequently used to describe experiences in which simulation is used as a tool for the testing process. In other works, the term “simulation” makes reference to a set of formulas that are solved by analytical means.

As an example of the first usage, in their collection of works, Lazić, Mastorakis and Velasĕvić [21] to [25] aim at raising awareness about the usefulness and importance of computer-based simulation in support of software testing. In their works, simulation is used to ease the design and execution of the testing processes of real military and defense systems.

Some analytical models of the software testing process can also be found. Zhang, Zhou, and Luo [26] propose a reward-Markov-chain-based quantitative model for sequential iterative processes and show how to use it to estimate the time for the software testing process. Similarly to this, Lizhi, Weiqin, Zhou and Zhang [27] propose an approach to model the testing process based on hierarchical time colored Petri Nets (HTCPN). However, while Petri-nets are good at modeling resources and parallel processing, simulation modeling models system components and their interactions, making it possible to conduct arbitrary time-related performance analysis, something which is not easy using Petri-nets.

Consequently, to overcome the problems of analytical methods, simulation modeling can be applied in the context of testing processes mainly because: a) it enables to find solutions when analytical methods fail; b) it is a more straightforward process than analytical modeling since the structure of the simulation model naturally mimics the structure of the real system, and c) it is scalable, flexible, and easy to communicate since the modeling tools use visual languages.

However, despite these advantages there is a small number of contributions of simulation modeling in the field of software testing processes. Saurabh [28] presents a System Dynamics (SD) model of software development with a special focus on the unit test phase. This work is partially based on Collofello’s et al. work about modeling the software testing process under the SD approach [29].

The motivation of these works is closely related to ours, but the models are built under a different simulation

approach. System Dynamics approach operates at high abstraction level and is mostly used for strategic modeling. Hence, since a simulation model can only be used at the abstraction level in which it has been created, such a highly abstract model is not adequate for the operational and tactical levels in which decision-making regarding the testing processes takes place. In our case, since our main interest is to simulate the testing processes the discrete event (DE) modeling, with the underlying process-centric approach, has been selected. Furthermore, we have also selected the agent-based (AB) approach to be used together with the discrete-event one resulting in a multi-paradigm simulation model.

Generally, each simulation approach (SD, DE, AB) provides a set of different abstractions. If the system being modeled is complex enough, and software development is, then it is preferable to integrate different simulation methods than using one single approach, since the final model will represent the real system more realistically.

When we used the search string ("multi-paradigm" OR "multi-method") AND "simulation" AND "software testing process" and others alike in the digital libraries and citation databases, no single work was retrieved. Therefore, given the results of the systematic literature review performed, not fully documented here for space reasons, to the best of our knowledge our proposal is the first one that aims at using multi-paradigm simulation modeling to improve decision making in software testing management.

III. MULTI-LAYER TEST PROCESS MODEL

Testing processes include a variety of management and technical activities that are organized in a process model in part 2 of the ISO standard for software testing: ISO/IEC/IEEE 29119-2:2013 [30]. The purpose of this international standard is to define a generic process model for software testing that can be used by any organization when performing any form of software testing. Testing is structured in a multi-layer process model that defines the software testing processes at (1) the organizational level, (2) test management level and (3) dynamic test level. More specifically, the dynamic test level describes how dynamic testing is carried out within a particular phase of testing (e.g., unit, integration, system and acceptance) or type of testing (e.g., performance testing, security testing and usability testing). It is composed of four processes that are depicted in Figure 1.

- Test Design & Implementation Process: Describes how test cases and test procedures are derived; these are normally documented in a test specification, but may be immediately executed.
- Test Environment Set-Up & Maintenance Process: Describes how the environment in which tests are executed is established and maintained.
- Test Execution Process: Describes how the test procedures generated as a result of the Test Design & Implementation Process are run on the test environment established by the Test Environment Set-Up & Maintenance Process.
- Test Incident Reporting Process describes how the reporting of test incidents is managed.

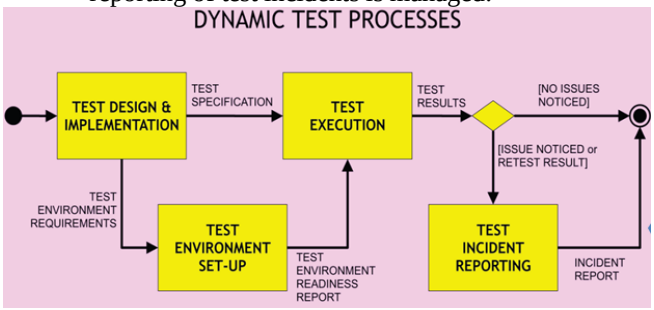


Figure 1. Dynamic Test Processes in ISO/IEC/IEEE 29119-2:2013.

The Test Execution Process is run after the tests have been specified and the environment has been established, which leads to a strong dependency on the previous processes. This process may need to be performed a number of times as all the available test procedures may not be executed in a single iteration. Additionally, this process must be reentered as a consequence of detected failures after the underlying defects have been corrected (retesting).

Besides, the Test Design & Implementation Process, and the Test Environment Set-Up & Maintenance Process may be reentered whether additional tests are needed after execution or some problems are detected in the testing environment. The Test Incident Reporting Process may be also reentered as a result of: a) the identification of test failures, b) something unusual or unexpected occurred during test execution, or c) retest activities.

IV. MODEL DESCRIPTION

The simulation model developed is described below in terms of its scope, result variables, process abstraction and input parameters. The description is organized following Kellner's proposal for describing simulation models [31].

A. Model Proposal and Scope

To determine a model proposal, the key questions that the model needs to address need to be identified. Then, model scope is set so that it is large enough to fully solve the key questions posed. In the context of this work, model proposal is to help in decision-making in software testing process management. Accordingly, the scope for this model will be a portion of the life cycle, with a short time span (i.e., the months in which the testing activities take place), one software product and two teams (i.e., development and testing teams) organizational breadth.

B. Result Variables

The result variables are the information elements needed to answer the key questions regarding the purpose of the model. In our model, several process metrics have been identified to help us understand the simulated process capability. According to this, process metrics have been classified into effectiveness and efficiency process metrics.

Effectiveness process metrics measure the extent to which a process produces a desired result [32].

The following result variables fall into this category:

- Defect Closure Period (DCP). The longer a reported defect takes to go from discovery to resolution, the higher the project risk associated with the underlying defect. Unresolved defects may: a) delay testing, b) make development less efficient or c) prevent the delivery of the software to the final customers. DCP measures the difference between the time required to repair a defect and the time required to confirm the defect is repaired.
- Defect Open Count. This measure tracks the number of times a defect report is opened. When the report is first submitted this count is set to one. This count is incremented each time the same defect report is reopened due to a failure in the confirmation test (retest).
- FixBacklog: Shows the percentage of defects closed per all the defects opened in a given time.
- Average Defect Life: Shows the average elapsed time from the moment a defect is found until it is successfully closed.
- Total Planned Test: The metric shows the evolution of the number of planned test cases along the testing project.
- Total Executed Tests: Shows the evolution of actual test cases that are executed along the project.
- Total Passed Tests: Shows the actual test cases that are executed and successfully passed (e.g., did not find any defects).
- Total Failed Tests: Shows the actual test cases that are executed and failed (e.g., did find defects).

Efficiency process metrics measure the extent to which a process produces its desired results in a not wasteful way and, ideally, minimizing the resources used [32].

Result variables in this category follow:

- Actual Test Time: Shows the total length of the testing process.
- Total Team Size and number of people per activity: Shows the total size of the testing team and the number of resources allocated to each activity of the process, respectively.
- Average Defect Cost: Shows the ratio between the total number of defects closed and the number of working hours invested.
- Process Efficiency: Shows the ratio of the number of defects closed per the number of defects found.

C. Process Abstraction

When developing a simulation model, the key elements of the process, their inter-relationships, and behavior need to be identified. The focus should be on those aspects of the process that are especially relevant to the purpose of the model, and believed to affect the result variables [31].

One of the decisions that need to be made in this phase is the simulation paradigm that it is going to be used to build the model. A simulation paradigm is a general framework for mapping a real world system to its model. The choice of paradigm should be based on the system being modeled and the purpose of the modeling. When modeling complex systems, it is frequent that different parts of the system are most naturally modeled using different paradigms. In this case, a multi-paradigm model is built.

In order to build our model, the multi-paradigm approach has been selected. First, to model and simulate the dynamic testing processes, the paradigm selected has been the discrete-event or process centric approach. Under this approach, the system being modeled is considered as a process, i.e., a sequence of operations being performed across entities, and this makes this paradigm the most natural and adequate to build process simulation models. The model is specified graphically as a process flowchart, where blocks represent the operations to be done along the process.

Although a simulation model following this approach allows us to analyze the evolution of the testing activities, the resource consumption and the number of defects detected, it would be interesting to add an extra functionality to the simulation model allowing the user to track the life of every defect since it is found until it is closed. It is important to notice that to achieve this aim the level of abstraction used needs to be changed from process-centric to individual-centric. Agent-based modeling is a simulation approach that allows the modeler to build a model under a bottom-up perspective, that is, describing the behavior of individuals (e.g., agents) and, if needed, their interactions. Frequently, the behavior of an agent is formalized by means of a state chart-like diagram. Therefore, this approach seems to be most natural and adequate to describe the lifecycle of defects found during the testing phase. As a consequence, a multi-paradigm simulation model was our choice for our modeling problem.

In summary, the model consists of two connected models. A description of each of these models follows:

1) Discrete event model (DE).

The discrete event model represents the Dynamic Test Processes in ISO/IEC/IEEE 29119-2:2013 [30], previously described in Section III.

The development process produces two main artifacts that are the input for the testing processes:

1. The test basis, usually the software specification, which is modeled as a set of features.
2. The executable code that is to be exercised by the tests.

The availability of the test basis enables the execution of the Test Design & Implementation Process, which leads to a number of test cases. However, test cases are not ready to be executed until the test environment has been established (Test Design & Implementation Process) and the executable code released. Once the code is installed in the testing environment, the Test Execution Process can begin. Failed test cases are the input for the Test Incident Reporting Process and the results communicated to the development processes through the Agent-based model. Test execution reenters when previously detected defects have been fixed by development.

2) Agent-based model (AB).

During the software development process, each defect has a lifecycle in which it reaches different states. In order to simulate the different states that a defect reaches the agent-

based paradigm has been used. Under this approach, we formalize the defects found as agents and their behavior as a state chart that reflects the different states and transitions of defect lifecycle. A description of each state in which the agents can be follows:

- *New*: An agent reaches this state when a defect is reported by the tester for the first time and is yet to be approved.
- *Analyzed*: Once a defect is reported, the manager has to analyze it in order to approve it as a genuine defect, reject or defer it. The agent remains in this state during the time in which this activity takes place. When the activity is done, the information for deciding what to do with the defect is available, and so, the agent moves to the next state, which can be one of the following: a) *Rejected*: If a defect is found to be invalid, b) *Deferred*: If a defect is decided to be fixed in upcoming releases, and c) *Assigned*: If a defect is found to be valid and assigned to a member of the development team to fix it.
- *Fixed*: An agent moves to this state once the developer communicates the defect is fixed. The defect goes to the testing team for validation by injecting a task in the DE model to indicate that the test case that found this defect has to be executed again (retest). The result of this execution will determine the next state of the agent.
- *Closed*. If the tester finds that the defect is indeed fixed and is no more a cause of concern, the agent moves to the state Closed. Otherwise, if the defect is not fixed or partially fixed, the agent will go again to the state Assigned in which the work of a developer working on its fixing will be simulated again.

D. Input Parameters

The input parameters to include in the model largely depend upon the result variables desired and the process abstractions identified. Input parameters allow setting up different scenarios for simulation. The input parameters of the simulation model are the following:

- Software size: Size of the software product under development.
- FPA per Feature: Adjusted Functional Points per feature.
- Number of Test Cases per Feature: Number of test cases that need to be designed and executed per feature.
- Initial number of tasks in Environment Setup. Initial number of tasks that need to be done for the common and global environment setup.
- Estimated Time for Environment Setup. Time estimated to develop each environment setup task.
- Environment Setup Resources. Number of people allocated to the Environment Setup processes.
- Estimated Time for Test Design and Implementation. Time estimated to develop each task of the Test Design and Implementation processes.
- Test Design and Implementation Resources. Number of people allocated to the Test Design and Implementation process.
- Estimated Time for Test Execution. Time estimated to develop each task of the Test Execution processes.
- Test Execution Resources. Number of people allocated to the Test Execution processes.
- Estimated Time for Test Incident Reporting. Time estimated to develop each task of the Test Incident Reporting processes.
- Test Incident Reporting Resources. Number of people allocated to the Test Incident Reporting Processes.
- Estimated time to fix a defect. Time estimated to a fix a defect by a developer.
- Code released for Test Execution. Indicates when the code is released for testing. This value is provided as a percentage of delay measured regarding the initial estimated time for the testing project.
- Probability of finding a defect per Test Case Execution. Probability that a Test Case finds a defect when the test case is executed the first time.
- Probability of finding a defect per Test Case in Retest Execution. Probability that a Test Case finds a defect when the defect has been reported as fixed.

In order to achieve more realistic results, the model accepts a triangular distribution for most of the above input parameters.

V. SIMULATION OPTIMIZATION

Even though simulation runs are useful to visualize the effect of different values of the input parameters in the process performance, that is, to execute what-if scenarios in managerial decision-making, a key benefit can be obtained when we use together simulation and metaheuristic optimization algorithms in a process called simulation optimization. In this case, it is possible to obtain which values need to take the input parameters in order to maximize or minimize an output variable.

This section presents two optimization scenarios regarding the following exploratory questions:

- RQ1: Is it possible to maximize the efficiency of the test process by controlling the moment in which the executable code is available for testing? The optimization will determine the distribution of the human resources that maximizes the Process Efficiency.
- RQ2: Is it possible to minimize the time life span of a defect? (time from detection to closing). The optimization will determine the distribution of the human resources that minimizes the Average Defect Life.

The model implementation and the simulation runs have been performed using Anylogic™ software [33] with the Enterprise Library. The model logic is written in Java. Optimizations have been carried on using the optimizer OptQuest® [34] built-in Anylogic™.

TABLE I. BASE SCENARIO CONFIGURATION

Input parameter	Value
Software size	800 FPA
FPA per Feature	5
Number of Test Case per Feature	(0.5, 2, 4)
Initial number of tasks in Environment Setup	5 tasks
Estimated Time for Environment Setup	(10, 14.4, 20) hours
Environment Setup Resources	1 person
Estimated Time for Test Design and Implementation	(3, 4.5, 6) hours
Test Design and Implementation Resources	4 people
Estimated Time for Test Execution	(1.5, 3.2, 4.5) hours
Test Execution Resources	4 people
Estimated Time for Test Incident Reporting	(1.5, 3, 4.5) hours
Test Incident Reporting Resources	1 person
Estimated time to Fix a defect	(3, 4.5, 6) hours
Code released for Test Execution	15%
Probability of finding a defect per Test Case Execution	(5%, 15%, 25%)
Probability of finding a defect per Test Case in ReTest Execution	(10%, 20%, 30%)

The first step will be to configure a base scenario. Then optimizations will be determined starting from this scenario.

A. Base Scenario Setup

In this scenario, the base simulation is run to determine the values of the result variables and analyze the results of the process. In order to obtain a set of reasonable parameters, we have estimated the costs of the different activities using a set of ratios observed in average risk profiles [35]. We consider functional testing for a system test phase in a project with waterfall development, experienced builders and a structured test approach driven by risk:

- Development process ratios: Ratios of functional design, realization and functional test are 1:2:1.
- Test process ratios: the ratios of test design & implementation, execution, reporting and environment set-up are 50:40:5:5, respectively.

The values of the input parameters in this scenario are displayed in Table I.

B. Base Scenario Run

Once the input parameters of the model have been set to the values shown in Table I, the model is ready to simulate the base scenario. The number of test cases in each state is depicted in Figure 2, which shows that initially 160 test cases were planned for the initial features. At the end of the simulation the total of fulfilled tests is 511 with 416 passed tests (81.41%) and 95 failed tests (18.59%).

Figure 3 depicts the number of defects in each state. At the end of the simulation, 4 of them were rejected and 3 of them deferred; 75 defects were closed and 9 reopened. The Process Efficiency reached with this setting is 91.0%, which is reasonable in practice, showing the consistency of the model when using the above parameters.

Figure 4 and Figure 6 display the time evolution of the number of test cases in each state and the number of defects in each state until the end of simulation, respectively. These figures are included later in the article to facilitate the comparison against the optimization runs. The increasing of the number of test cases in each states (Figure 4) is fairly linear from the moment in which testing begins. The number of defects in each state (Figure 6) follows a different trend, as there is a significant delay from detection of failures to their closing. This is related to the Average Defect Life, which will be optimized later.

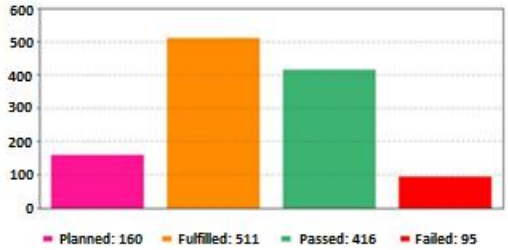


Figure 2. Number of test cases in each state at the end of the simulation.

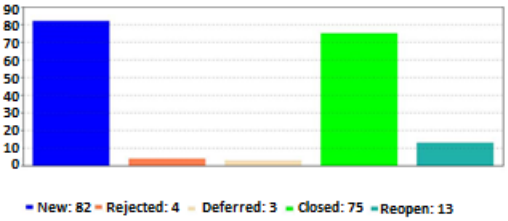


Figure 3. Number of defects in each state at the end of the simulation.

Figure 12 displays the time evolution of the Process Efficiency, which has been defined before as the ratio between the number of closed defects and the number of defects found. At the beginning of the simulation the number of defects found is zero (because test cases are still in preparation), so that the simulator returns 100%. Just after the first test case is available, the efficiency goes to zero as there are not closed defects. After the first defect has been closed, efficiency increases up and grows towards its final value (91.0%).

C. Optimization of the Process Efficiency

To answer RQ1, we ran an optimization experiment to determine whether it is possible to improve the efficiency of the test process by controlling the moment in which the executable code is available for testing. The optimization will determine the distribution of the human resources that maximizes test efficiency when the code is released for testing in a range that varies from 5% to 50% from the moment the testing process begin [1]. Table II displays the input values for the control parameters of the experiment, the constraints imposed and the results obtained in the optimized process compared with the base case.

The results of the optimization experiment show that, under the constraints imposed, it is possible to achieve 97% of efficiency in the process allocating 7 people to the process and having a maximum delay of the code released for testing of 27% of the initial estimated time. This will result into a process that is 97% efficient in closing defects but finishes one month later than the base scenario.

TABLE II. OPTIMIZATION OF THE PROCESS EFFICIENCY COMPARED WITH BASE SCENARIO

Input parameter	**Control Input**	**Result (Base scenario)**	**Result (Optim. scenario)**
Initial number of tasks in Environment Setup	3-5 tasks	5	5
Environment Setup Resources	1-4 people	1	1
Test Design and Implementation Resources	1-4 people	4	2
Test Execution Resources	1-4 people	4	3
Test Incident Reporting Resources	1-4 people	1	1
Code released for Test Execution	5% - 50%	15%	27%
Constraints	**Value**		
Testing Team Size	<= 7 people		
Maximum Testing Time Overrun	<= 1 month		
Process Efficiency obtained (percent)		90%	97%

TABLE III. OPTIMIZATION OF THE AVERAGE DEFECT LIFE COMPARED WITH BASE SCENARIO

Input parameter	**Control Input**	**Result (Base scenario)**	**Result (Optim. scenario)**
Initial number of tasks in Environment Setup	3-5 tasks	5	5
Environment Setup Resources	1-4 people	1	1
Test Design and Implementation Resources	1-4 people	4	2
Test Execution Resources	1-4 people	4	4
Test Incident Reporting Resources	1-4 people	1	3
Code released for Test Execution	5% - 50%	15%	20%
Constraints	**Value**		
Testing Team Size	<= 10 people		
Maximum Testing Time Overrun	<= 1 month		
Average Defect Life (working hours)		28.31	13.91

The conclusion drawn from this particular experiment with regard to the base scenario is that if the project is adequately scheduled, it is possible to reduce the total number of test resources as well as increase the process efficiency.

D. Optimization of the Average Defect Life

To answer RQ2 we run an optimization experiment to determine whether it is possible to improve the time span between fault detection and closing by controlling the moment in which the executable code is available for testing (in an range that varies from 5% to 50% as in previous subsection). In this case, the optimization will minimize the Average Defect Life. Table III displays the input values for the control parameters of the experiment, the constraints imposed and the results obtained in the optimized process compared with the base case.

The results of the optimization experiment show that under the constraints imposed, it is possible to reduce by more than a half (down to 13.91%) the Average Defect Life by allocating the same amount of people in a different way to the process and having a maximum delay of the code released for testing of 20% of the initial estimated time.

As in the previous optimization, this case also requires a team size of 10 people allocated to the testing tasks. However, the optimization brings new light regarding the allocation of people to the tasks resulting in a considerable advantage regarding the average defect life.

Now, a comparison on trends of the main variables of the process will be provided. Figure 4 and Figure 5 display the time evolution of the number of test cases in each state for the base and optimized scenarios, respectively. Figure 6 and Figure 7 display the time evolution of the number of defects in each state until the end of simulation for the base and optimized scenarios, respectively. In Figure 7, it can be seen that there is a shorter delay between the moment in which failures are detected and their closing, at the expenses of a larger test time.

Figure 8 and Figure 9 display the values of the Average Defect Life (base and optimized scenario, respectively). In the optimized scenario, the variable starts growing earlier than in the base scenario, but with a lower maximum value. Just after reaching the maximum begins a continuous decrease until its optimum value (13.9 working hours) is achieved, a much lower valued than the corresponding value for the base scenario (127.4 working hours). The Average Defect Cost (ratio between number of defects closed and total time spent) is displayed in Figure 10 and Figure 11, showing similar trends and final values of 6.2 working hours (base scenario) and 5.6 hours (optimized scenario).

To finish, a comparison of the Process Efficiency is provided in Figure 12 and Figure 13. Process Efficiency at the end (89.0%) is marginally lower than in the base scenario (90.0%), since this optimization is intended to minimize the average defect life, but it begins growing at earlier stages of the testing project.

Other simulations can help find the best input values for project schedule, resource allocation and quality objective from among all that lead to the optimization of the key process outputs. Moreover, the results of optimizations presented in this paper have been performed separately, but this makes room for future explorations in multi-objective optimizations. For example, in order to balance the maximization of variables like Process Efficiency as well as the minimization of variables like Average Defect Life.

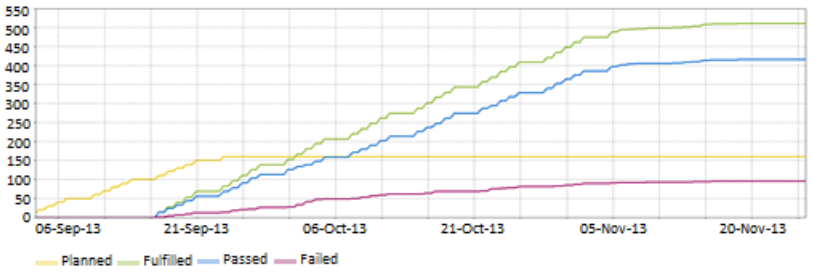


Figure 4. Time evolution of the number of test cases in each state (base scenario).

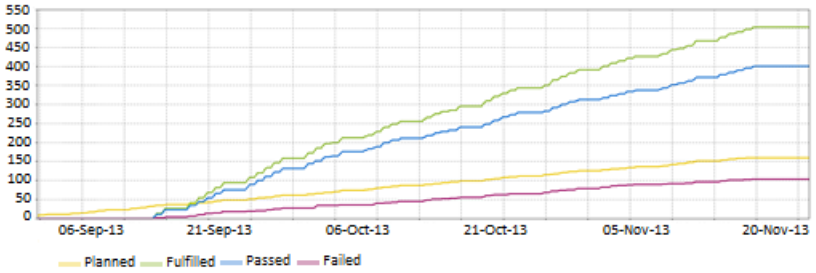


Figure 5. Time evolution of the number of test cases in each state (optimized scenario).

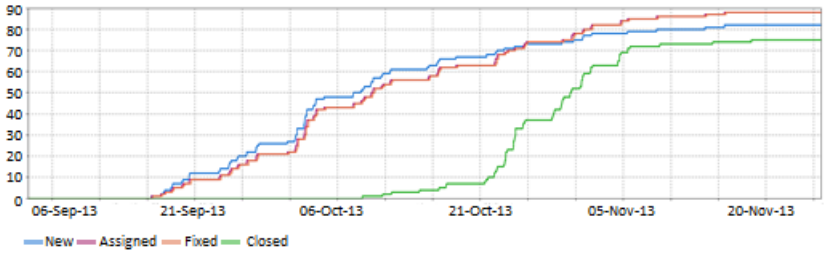


Figure 6. Time evolution of the number of defects in each state (base scenario).

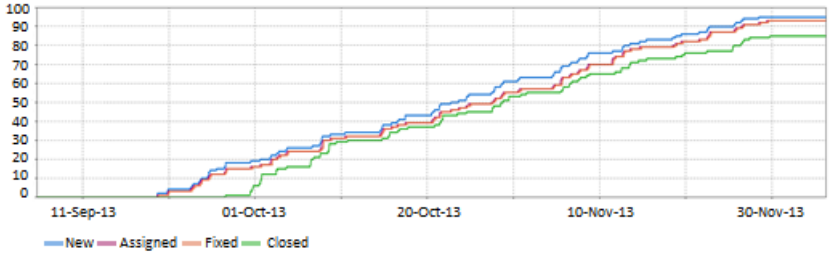


Figure 7. Time evolution of the number of defects in each state (optimized scenario).

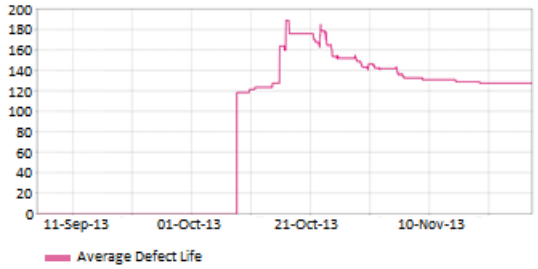


Figure 8. Time evolution of average defect life (base scenario).

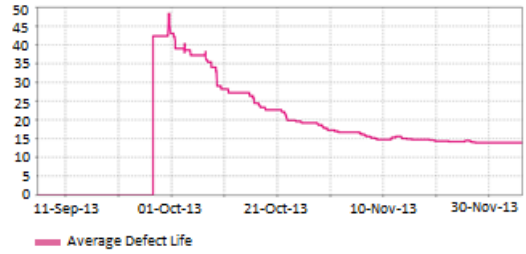


Figure 9. Time evolution of the average defect life (optimized scenario).

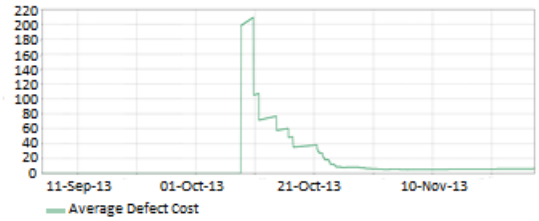


Figure 10. Time evolution of the average defect cost (base scenario).

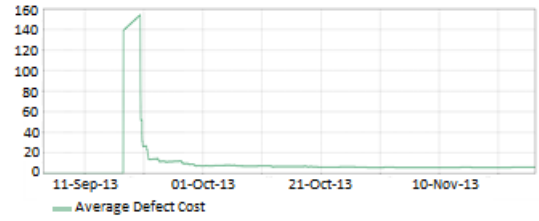


Figure 11. Time evolution of the average defect cost (optimized scenario).

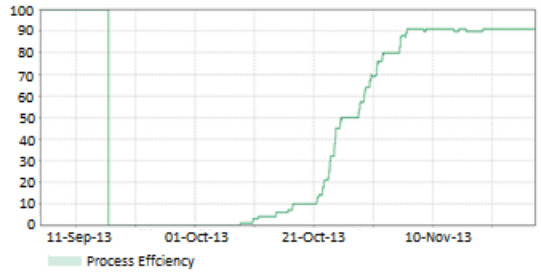


Figure 12. Time evolution of the process efficiency (base scenario).

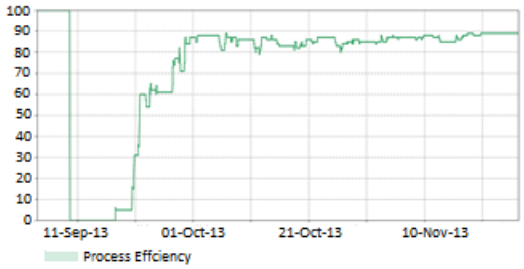


Figure 13. Time evolution of the average process efficiency (optimized scenario).

VI. Conclusion and Further Work

This paper presented a simulation model for the dynamic testing processes that allows a seamless integration between the testing and development processes. The model is devised as a multi-paradigm model composed by a discrete event simulation model, to simulate the execution of the dynamic test processes, and an agent-based simulation model, to in-depth simulate the defects life cycle. The model has been first used to simulate a base scenario. The results of the simulation runs were then used to design two simulation optimization scenarios. By merging simulation and optimization it is possible to use the model to find the best testing team configuration so that key process metrics are optimized. Results show that the simulation model can be effectively used to optimize different process metrics (Test Process Efficiency and Average Defect Life) and then help managers to achieve a trade-off between cost, schedule and quality.

This work is a first step in the use of multi-paradigm simulation models for testing management. Further work will include, although not limited to, the consideration of agent-based models to simulate parts of the dynamic test processes, the integration into a more complex project development simulation model [36], multi-objective optimization and experimentation in different projects using different lifecycle models and including different test levels of testing. After calibrating and validating the model with historical data from the industry, it will be also possible to exploit it as an operating tool for decision-making in the industrial domain.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Science and Technology with ERDF funds under grants TIN2010-20057-C03-03, TIN2010-20057-C03-01, TIN2013-46928-C3-2-R and TIN2013-46928-C3-1-R.

REFERENCES

[1] M. Ruiz, J. Tuya, and D. Crespo, "Simulation-based management for software dynamic testing processes," Proceedings of the 7th International Conference on Software Engineering Advances (ICSEA 2012), IARIA 2012, Lisbon, pp. 630-635.

[2] E. van Veenendaal (ed), Standard glossary of terms used in software testing, Version 2.1, International Software Testing Qualifications Board, Oct. 2010.

[3] RJ. Madachy, Software process dynamics. John Wiley & Sons, Inc., 2008.

[4] K. Nogeste and DHT. Walker, "Using knowledge management to revise software-testing processes," Journal of Workplace Learning, 2006, 18(1), pp. 6-27.

[5] R. Abdullah, ZD. Eri, and AM. Talib, "A model of knowledge management system in managing knowledge of software testing environment," 5th. Malaysian Conference in Software Engineering, (MySEC 2011), 2011, pp. 229-233.

[6] J. Andrade, J. Ares, M. Martínez, J. Pazos, S. Rodríguez, J. Romera, and S. Suárez, "An architectural model for software testing lesson learned systems," Information and Software Technology, 2013, vol. 55, no. 1, pp, 18-34.

[7] X. Liu, G. Gu, Y. Liu, and J. Wu, "Research and implementation of knowledge management methods in software testing process," WRI World Congress on Computer Science and Information Engineering, (CSIE 2009), 2009, pp. 739-743.

[8] Z. Bluvband, S. Porotsky, and M. Talmor. "Advanced models for software reliability prediction," Proceedings - Annual Reliability and Maintainability Symposium, 2011, pp. 1-5.

[9] L. Lazić and N. Mastorakis, "The COTECOMO: COnstractive test effort COst Model," N. Mastorakis and V. Mladenov (Eds) Proceedings of the European Computing Conference, vol. 2, Series: Lecture Notes in Electrical Engineering, 2009, vol. 27, pp. 89-110.

[10] SD. Kanawat, A. Pandey, A. Singh, and A. Maloo, "Software testing model for quality," Advanced Materials Research, 2001, vol. 4507, pp. 403-408.

[11] L. Xin-Ke and Y. Xiao-Hui, "A goal-driven measurement model for software testing process," WRI World Congress on Software Engineering, (WCSE 2009), 2009, vol. 4, pp. 8-12.

[12] L. Lazić, "Software testing optimization by advanced quantitative defect management," Computer Science and Information Systems, 2010, 7(3), pp. 459-487.

[13] J. Saldaña-Ramos, A. Sanz-Esteban, J. García-Guzmán, and A. Amescua, "Design of a competence model for testing teams," IET Software, 2012, 6(5), pp. 405-415.

[14] A. Farooq, K. Georgieva, and RR. Dumke, "A meta-measurement approach for software test processes," Proceedings of the 12th. IEEE International Multitopic Conference (IEEE INMIC 2008), 2008, pp. 333-338.

[15] X. Li and W. Zhang, "The PDCA-based software testing improvement framework," Proceedings of the 2010 International Conference on Apperceiving Computing and Intelligence Analysis, (ICACIA 2010), 2010, pp. 490-494.

[16] L. Han, "Evaluation of software testing process based on bayesian networks," Proceedings of the 2010 International Conference on Computer Engineering and Technology, (ICCET 2010), 2010, 7, pp. V7361-V7365.

[17] D. Zhang, C. Nie, and B. Xu, "Cross-entropy method based on Markov decision process for optimal software testing," Ruan Jian Xue Bao/Journal of Software, 2008, vol. 19, no. 10, pp. 2770-2779.

[18] Q. Li, Y. Yang, M. Li, Q. Wang, BW. Boehm, and C. Hu, "Improving software testing process: Feature prioritization to make winners of success-critical stakeholders," Journal of Software: Evolution and Process, 2012, vol. 24, no. 7, pp. 783-801.

[19] K. Cai, Z. Dong, and K. Liu, "Software testing processes as a linear dynamic system," Information Sciences, 2008, vol. 178, no. 6, pp. 1558-1597.

[20] HKN. Leung, "Improving the testing process based upon standards," Software Testing Verification and Reliability, 1997, vol. 7, no. 1, pp. 3-18

[21] L. Lazić and N. Mastorakis, "RBOSTP: Risk-based optimization of software testing process. Part 1," WSEAS Transactions on Information Science and Applications, 2005, vol. 2, no. 6, pp. 695-708.

[22] L. Lazić and N. Mastorakis, "RBOSTP: Risk-based optimization of software testing process. Part 2," WSEAS Transactions on Information Science and Applications, 2005, vol. 2, no. 7, pp. 902-916.

[23] L. Lazić and N. Mastorakis, "Integrated intelligent modeling, simulation and design of experiments for Software Testing Process," Proceedings of the International Conference on Computers, 2010, vol. 1, pp. 555-567.

[24] L. Lazić and N. Mastorakis, "The use of modeling & simulation-based analysis & optimization of software testing," WSEAS Transactions on Information Science and Applications, 2005, vol. 2 no. 11, pp. 1918-1933.

[25] L. Lazić and D. Velasĕvić, "Applying simulation and design of experiments to the embedded software testing process," Software Testing Verification and Reliability, 2004, vol. 14, no. 4, pp. 257-282.

[26] WM. Zhang, BS. Zhou, and WJ. Luo, "Modeling and simulating of sequential iterative development processes," Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, (CIMS 2008), 2008, vol. 14, no. 9, pp. 1696-1703.

[27] C. Lizhi, T. Weiqin, B. Zhou, and J. Zhang J, "Modeling software testing process using HTCPN," Fourth International Conference on Frontier of Computer Science and Technology, (FCST 2009), 2009, pp. 429-434.

[28] K. Saurabh, "Modeling unit testing processes: a system dynamics approach," Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2008), 2008, vol. ISAS 1, pp. 183-186.

[29] JS. Collofello, Y. Zhen, JD. Tvedt, D. Merrill, and I. Rus, "Modeling software testing processes," Proceedings of the International Phoenix Conference on Computers and Communications, 1996, pp. 289-293.

[30] ISO/IEC/IEEE 29119-2:2013 Software and Systems Engineering - Software Testing – Part 2: Test processes. August 2013.

[31] MI. Kellner, RJ. Madachy, and DM. Raffo, "Software Process Modeling and Simulation: Why, What, How?," Journal of Systems and Software, April, 1999, Vol. 46, no. 2/3.

[32] R. Black, "Managing the testing process: practical tools and techniques for managing hardware and software testing," Wiley Publishing, 2002.

[33] XJ Technologies. Anylogic™. http://www.anylogic.com/ [retrieved: May, 2014] .

[34] OpTek Systems, Inc. OptQuest®. http:// www.opttek.com/ [retrieved: May, 2014].

[35] T. Koomen, L. van der Aalst, B. Broekman, and M. Vroon, "TMap Next for result-driven testing," UTN Publishers, 2007.

[36] D. Crespo and M. Ruiz, "Decision making support in CMMI process areas using multiparadigm simulation modeling," 2012 Winter Simulation Conference (WSC 2012), 2012, pp. 1-12.

Implementation Variants for Position Lists

Andreas Schmidt*†, Daniel Kimmig†, and Steffen Scholz†
* *Department of Computer Science and Business Information Systems,*
Karlsruhe University of Applied Sciences
Karlsruhe, Germany
Email: andreas.schmidt@hs-karlsruhe.de
† *Institute for Applied Computer Science*
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: {andreas.schmidt, daniel.kimmig, steffen.scholz}@kit.edu

***Abstract*—Within "traditional" database systems (*row store*), the values of a tuple are usually stored in a physically connected manner. In a *column store* by contrast, all values of each single column are stored one after another. This orthogonal storage organization has the advantage that only data from columns which are of relevance to a query have to be loaded during query processing. Due to the storage organization of a *row store*, all columns of a tuple are loaded, despite the fact that only a small portion of them are of interest to processing. The orthogonal organization has some serious implications on query processing: While in a traditional *row store*, complex predicates can be evaluated at once, this is not possible in a column store. To evaluate complex conditions on multiple columns, an additional data structure is required, the so-called *Position List*. At first glance these *Position Lists* can easily be implemented as a dynamic array. But there are a number of situations where this is not the first choice in terms of memory consumption and time behavior. This paper will discuss some implementation alternatives based on (compressed) bitvectors. A number of tests will be reported and the runtime behavior and memory consumption of the different implementations will be presented. We additionally extended the existing WAH library for compressed bitvectors by a number of new methods to be used for the purpose of implementing the functionality of *Position Lists* based on (compressed) bitvectors. Finally, some recommendation will be made as to the situations in which the different implementation variants for *Position Lists* will be suited best. Their suitability depends strongly on the selectivity of a query or predicate.**

Keywords–Column stores; PositionList implementation variants; bitvector; compression; run length encoding; performance measure

I. Introduction

This article is an extended version of a paper entitled *Considerations about Implementation Variants for Position Lists* [1] presented at the Fourth International Conference on Advances in Databases, Knowledge, and Data Applications in Sevilla, Spain. Some important extensions include the extension of the WAH library by a number of transformation functions between different representation forms of a *Position List* as well as the implementation of a special append-based bitset function on compressed bitvectors, which allows the usage of compressed bitvectors in the typical *Position List* generation process when evaluating a single predicate.

Nowadays, modern processors utilize one or more cache hierarchies to accelerate access to main memory. A cache is a small and fast memory which resides between the main memory and the CPU. In case the CPU requests data from the main memory, it is first checked, whether these data are already contained in the cache. In this case, the item is sent directly from the cache to the CPU, without accessing the much slower main memory. If the item is not yet in the cache, it is first copied from the main memory to the cache and then sent to the CPU. However, not only the requested data item, but a whole cache line, which is between 8 and 128 bytes long, is copied into the cache. This prefetching of data has the advantage of requests to subsequent items being much faster, because they already reside within the cache. Depending on the concrete architecture of the CPU, the speed gain when accessing a data set in the first-level cache is up to two orders of magnitude compared to regular main memory access [2]. This means that when a requested data item is already in the first-level cache, the access time is much faster compared to the situation, when the data item must be loaded from the main memory (this situation is called a cache miss). The use of special data structures which increase cache locality (the preferred access of data items already residing in the cache) is called *cache-conscious* programming.

Column stores take advantage of this prefetching behavior, because values of individual columns are physically connected. Therefore, they often already reside in the cache when requested, as the execution of complex queries is processed column by column rather than tuple by tuple. This difference between a "traditional" *row store* and a *column store* is illustrated in Figure 1. In the upper part of the figure, a relation, consisting of six tuples, each with five columns, is shown. The lower part of the figure shows the physical layout of this relation on disk or in the main memory. On the left side, the row store layout is represented. The row store stores all values of one tuple in a physically connected

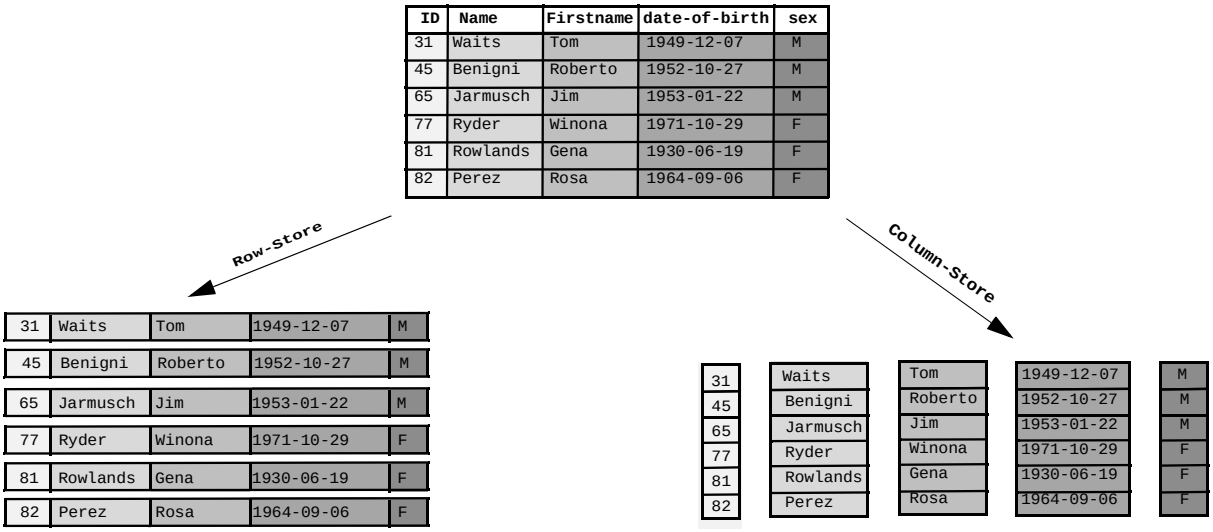


Figure 1. Comparison of the layouts of a row store and a column store (from [3])

manner. In contrast to this, a column store contains all values of each single column one after another.

This also means that the decision whether a tuple fulfills a complex condition on more than one column is generally delayed until the last column is processed. Consequently, additional data structures are required to administrate the status of a tuple in a query. These data structures are referred to as *Position List*s. A *Position List* stores information about matching tuples. The information is stored in the form of a Tuple IDentifier (TID). The TID is nothing more than the position of a value in a column. Execution of a complex query generates a *Position List* with entries of the qualified tuples for every simple predicate.

Complex predicates on multiple columns can be evaluated in two different ways. First, as shown in Figure 2, the predicates can be evaluated separately, and in a subsequent step, the resulting *Position Lists* can be merged. The advantage of this variant is, that the evaluation of the predicates can be done in parallel. A drawback of this solution is, that all column values must be evaluated.

In contrast to this, the evaluation of the query can also be done sequentially, as shown in Figure 3. In this case, a *Position List* representing the result of a previously evaluated predicate is an additional input parameter for the evaluation of the second predicate. Not all column values have to be evaluated, but only those for which an entry in the first *Position List* exists. The drawback of this solution is the strict sequential program flow and a slightly more complex execution, which may probably cause more cache misses compared to the parallel version. Which of the variants is better suited depends on the boundary conditions of the query.

In previous work, we developed the Column Store Toolkit (CSTK) [3] and used it as a starting point for further research

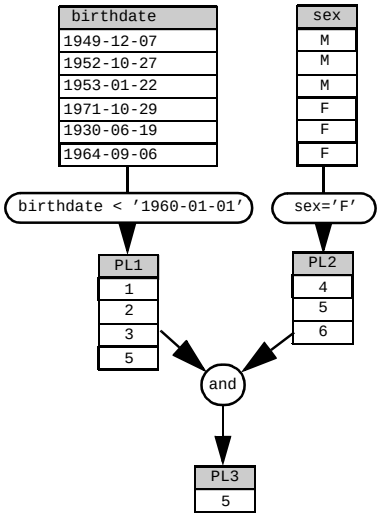


Figure 2. Isolated evaluation of predicates on their corresponding *Position Lists* and subsequent merging of the resulting *Position Lists* (from [3])

in the field of optimizing SQL queries based on a column store architecture [4].

The main objective of this paper is to present an in-depth analysis of different implementation variants of *Position Lists* and to demonstrate their advantages and disadvantages in different situations in terms of runtime behavior and memory consumption.

The paper is structured as follows. After giving an overview over related work in the next section, we will discuss some specific details of *Position Lists*. Then, the most important components of the CSTK will be introduced. After that, a number of experiments with respect to runtime behavior and memory consumption will be performed in

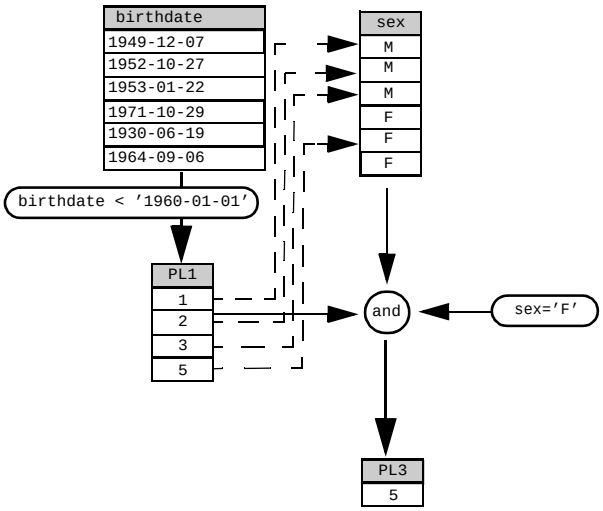


Figure 3. Iterative evaluation of predicates, using *Position Lists* as additional parameters

the main part. Finally, results will be summarized, and an outlook will be given on future research activities.

II. Related Work

First work addressing column stores (vertical storage orientation) is dated back into the 80th [5], [6]. During the last decade, a number of new research prototypes, based on a vertical partitioning of data, appeared and did show some advantages. From these systems, C-Store [7] and MonetDB [8] are the most notably ones. On the commercial side, Infobright [9], SAP-Hana [10], Sybase IQ [11], and Vertica [12] (a commercial version of C-Store), amongst others, appeared. Today, also big players like Oracle and Microsoft implemented columns store technologies into their database systems [13], [14]. Various publications compare the performance of column stores with that of row stores for different workloads [7], [15], [16]. In contrary, [17] examines different execution plan variants for column stores, while [18] considers the impact of compression. Following the work in [17], we examine different implementation variants for the underlying data structures and algorithms of the operations used in the execution plan of a query.

Abadi et. al. in [19] mention different implementation variants for *Position Lists*, i.e., a simple array, a bitstring or a set of ranges of positions, but did not compare these different solutions with respect to runtime behaviour. In contrast to the previous mentioned work, we do not use a fixed structure for implementing the *Position Lists*, but compare the runtime and memory consumption behaviour of different implementation variants with respect to the selectivity of a predicate.

III. Position Lists

From a logical point of view, a *Position List* is nothing more than an array or list with elements of the data type *unsigned integer* (UINT) as far as structure is concerned. However, it has a special semantics. The *Position List* stores TIDs. A *Position List* is the result of a query via predicate(s) on a *Column*, where the actual values are of no interest, whereas the information about the qualified data sets is desired. *Position List*s store the TIDs in ascending order without duplicates. In other words, a *Position List* stores the information for each tuple no matter whether it belongs to a result (so far) or not.

A. *Operations on* Position Lists

The fundamental logical operations on *Position Lists* are appending TIDs at the end (write operation), iterating over the list of TIDs (read operation), and performing *and/or* operations on complete lists.

Further operations that are mainly based on this basic functionality, include the materialization [17] of the corresponding values from the requested columns, the storage of the whole list or parts of it in a file, and the import from a file.

B. *Implementation Variants*

Based on the logical structure and behavior discussed above, the first intuitive implementation of a *Position List* is using a dynamic array (an array of flexible size) of unsigned integer values. The advantage of this variant is, that the implementation is straight forward and the storage of the TIDs is cache-conscious [20], [21] in the context of the above-mentioned operations like iterating, storing, and *and/or* operations.

As *Position List*s store the TIDs in ascending order without duplicates, typical *and/or* operations are very fast, as the cost for both operations is $O(|Pl_1| + |Pl_2|)$.

One big drawback of the implementation as a dynamic array is the fact, that the lists may be very large. This is especially true for predicates over multiple columns, where no predicate has a *high selectivity*. In this context, *high selectivity* means, that only a small number of tuples qualify the condition. A *low selectivity*, by contrast, means that a lot of tuples satisfy the condition. Typical predicates of low selectivity are the "family status" or the "gender" of a person. Let us consider a conjunctive query consisting of 6 predicates on different columns. Each single predicate has a selectivity of up to 50% (i.e., gender, family status, etc.). The overall selectivity of the query is about 1.5% of the original number of tuples, but the size of the cardinality of the individual *Position Lists* is up to 50% of the original table. Starting with the predicate of the highest selectivity and iteratively examining the values of tuples from the subsequent columns which qualified previously (see Figure 3) can reduce this problem. However, if no or only vague

information about the selectivity of the different predicates is available, this can be a serious problem. Figure 4 shows the size of a *Position List* in megabytes with respect to the selectivity of the predicate for a 100 million tuple table. In the worst case, the resulting *Position List* can be bigger than the original column (e.g., for columns with binary values or a small number of possible values only).

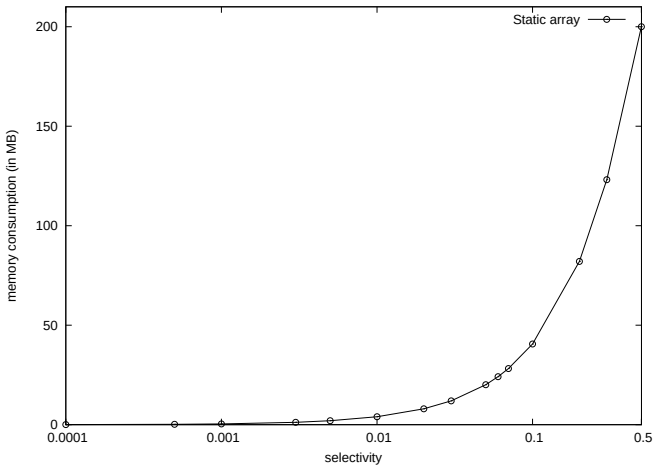


Figure 4. Memory consumption of a *Position List* implemented as array (logarithmic scale on x-axis)

The problem of the unpredictable size of the intermediate *Position Lists* can be prevented by using a bitvector to represent the *Position List*. Here, every tuple is represented by one bit. A value of '1' means that the tuple belongs to the (intermediate) result, a value of '0' means that the tuple does not belong to the result.

This has the advantage of the size of a *Position List* being exactly predictable, independently of the selectivity of the predicate. The selectivity only has impact on how many bits are set to '1'. Moreover, the two important operations *and* and *or* can be mapped on the respective primitive processor commands, which makes the operations fast. If *Position List*s are sparse, bitvectors can also be compressed very well using run length encoding (RLE) [22]. The idea behind RLE encoding is that if only a small number of bits are one, the '0' bits are not stored physically, but only the number of '0' bits are stored.

Figure 5 presents a simple example of this principle.

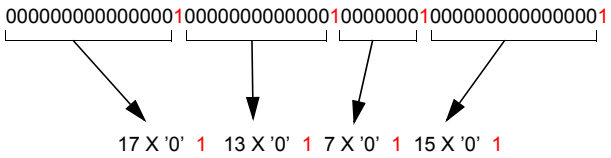


Figure 5. Principle of run length encoding (RLE)

The *Word Aligned Hybrid* algorithm (WAH) [23] uses this principle and distinguishes between two word types: *fills* and *literals*. The two word types are distinguished by the most significant bit, so 31 (63) bits remain for the stored bits per word or the length field. A *literal* is a word consisting of 31 (63) bits, of which at least one bit is '1'. A *0-fill* consists of a multiple of 31 (63) '0' bits which are stored in one word. The maximum number of '0' bits which can be stored in one word is $31 * 2^{31}$ (resp. $63 * 2^{63}$ for the 64-bit version).

The necessary operations like iterating, *and*, *or* can be performed on the compressed lists, thus avoiding a temporary decompression of the compressed representation. In the context of this paper, the bitvector implementation of the WAH algorithm and a simple plain uncompressed bitvector implementation are used. The WAH algorithm is considered to be one of the fastest algorithms for performing logical *and/or* operations on compressed bitvectors.

IV. The Column Store Toolkit

The Column Store Toolkit (CSTK) was deveolped as a toolkit with a minimum amount of basic components and operations required for building column store applications. These basic components were used as catalysts for further research into column store applications and for building data-intensive, high-performance applications with minimum expenditure.

The main focus of our components is on modeling the individual columns, which may occur both in the secondary store as well as in main memory. Their types of representation may vary. To store all values of a column, for example, it is not necessary to explicitly store the TID for each value, because it can be determined by its position (dense storage). To handle the results of a filter operation, however, the TIDs must be stored explicitly with the value (sparse storage).

Another important component is the already discussed *Position List*. Just like columns, two different representation forms are available for main and secondary storage. In this paper, it is concentrated on the main memory behavior of the *Position Lists*.

To generate results or to handle intermediate results consisting of attributes of several columns, data structures are required for storing several values (so-called multi-columns). These may also be used for the development of hybrid systems as well as for comparing the performance of row and column store systems.

The operations mainly focus on writing, reading, merging, splitting, sorting, projecting, and filtering data. Predicates and/or *Position List*s are applied as filtering arguments.

Figure 6 presents an overview of the most important operations and transformations among the components. The arrows show the operations among the different components (ColumnFile, Dense-/Sparse ColumnArray, PositionList, and PositionListFile). For a detailed description of the operations, see [3].

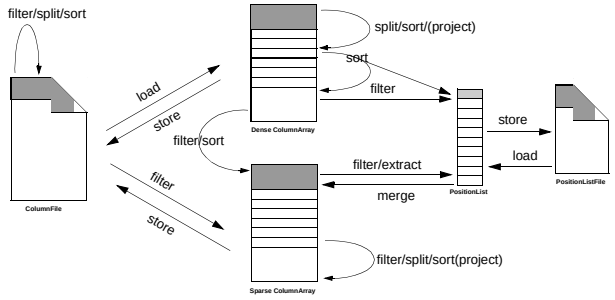


Figure 6. CSTK: Components and operations (from [3])

V. MEASUREMENTS

A. Elemental Operations

1) Experimental setup: All tests were performed on a 64-bit laptop running Windows 7 Enterprise with an Intel(R) Core(TM) i7-3520M CPU @ 2 x 2.90 GHZ and 8 GB of RAM. The used C++ compiler was gcc 4.5.3.

2) Memory consumption: In a first experiment, we compare the size of the different data structures with respect to memory consumption. As shown in Figure 7, the behavior of the array implementation is quite good for very high selectivities (0.01 and below), but changes for the worse at medium and low selectivities. Uncompressed bitvectors (plain bitvector, WAH-uncompressed) behave independently for all selectivities, their size is determined by the number of tuples in a table only. Compressed bitvectors show a very good behavior for all selectivities. If the selectivities get low, they behave like uncompressed bitvectors (compared to a pure uncompressed implementation of a bitvector, there will be a slight overhead of 1/32 resp. 1/64.). From a selectivity of about 3%, the array has a higher memory consumption than the uncompressed bitvector.

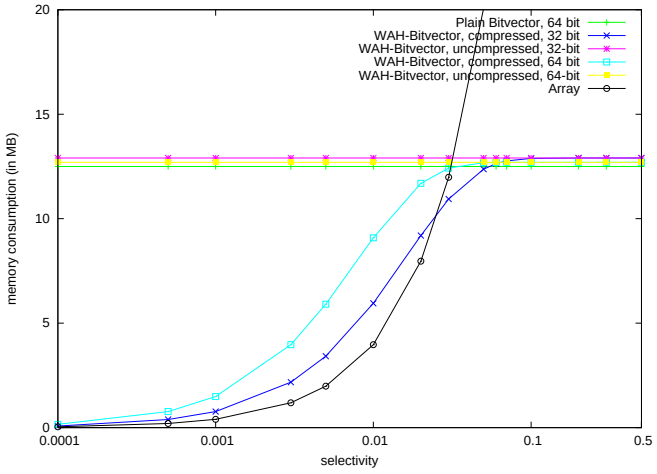


Figure 7. Memory consumption of different implementation alternatives for *Position Lists*

3) Iterating over TIDs: In the next experiment, we examine the runtime behavior of the two elemental operations:

- Appending TIDs on a *Position List*
- Iterating over the TIDs in a *Position List*.

These two operations are heavily used in the implementation of the CSTK components.

We implement a simple bitvector class on our own (without compression facility) and also use the well-known WAH algorithm. The overhead of the uncompressed representation of WAH is quite small in terms of both runtime and memory consumption.

In contrast to the original implementation of the WAH algorithm, we also use hardware support for special operations. The Leading Zero Count Instruction (LZCNT) is used to find the '1' bits inside a processor word. This leads to a performance advantage of a factor of 3 compared to the orginal WAH version.

In our first experiment, we take a table of 100 million tuples and formulate predicates with different selectivities between 0.0001 and 0.5. The TIDs of the qualified tuples are then stored in the different representation forms (plain bitvector, WAH bitvector uncompressed/compressed with 32 and 64 bit word size, array). After that, we measure the time to iterate over all the stored TIDs.

Figure 8 presents an overview of the runtime behavior for our different implementations:

The fastest implementation for all selectivities is the dynamic array. In contrast to this, the worst runtime behavior is reached by the standard WAH iterator (both 32- and 64-bit version), which therefore will not be considered any further. More interesing values come from the iterators which use the *__builtin_clzl* instruction from the gnu compiler family, which is mapped on the LZCNT instruction, if available (the plain bitvector implementation is the fastest).

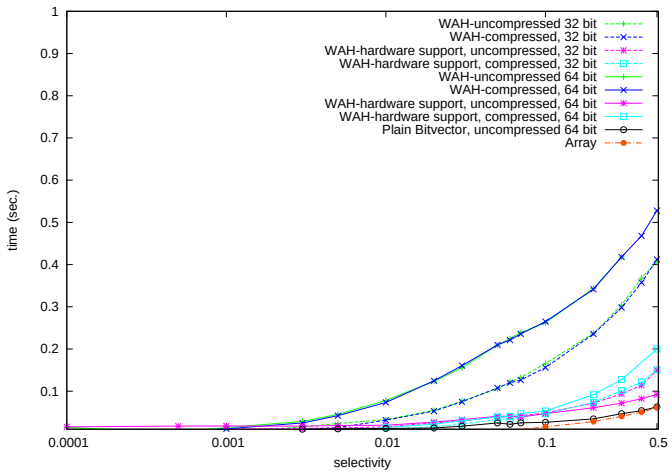


Figure 8. Measured time to iterate over 100 million data sets with different selectivities

Two more detailed graphs are given in Figure 9 and Figure 10. Here the static array implementation and the LZCNT-supported iterators are considered for high and low selectivity, respectively.

While Figure 9 shows the details for selectivities between 0.0001 and 0.05, Figure 10 shows the lower selectivities between 0.05 and 0.5. One interesting point is, that with low selectivity (Figure 10) the hardware-supported iteration behaves differently for the 32- and 64-bit WAH version. While the compressed version is faster for the 32-bit version, the opposite is true for the 64-bit version. This behavior can be found with the better compression ratio of the 32-bit version for lower selectivities, which leads to a smaller amount of memory which has to be loaded into the CPU.

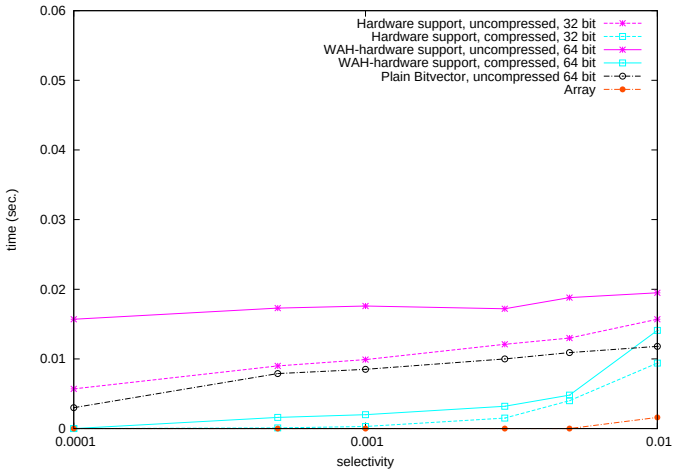


Figure 9. Measured time to iterate over 100 million data sets with high selectivity

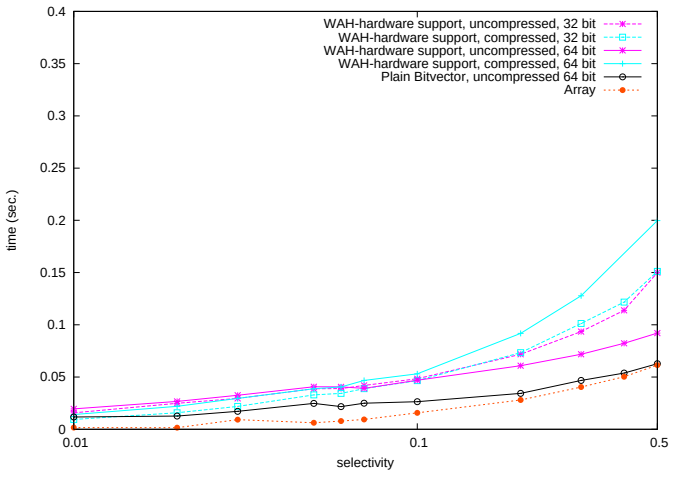


Figure 10. Measured time to iterate over 100 million data sets with low selectivity

It is obvious that the time for the uncompressed bitvector versions is the least dependent on the selectivity. This can be explained by the dominating time for loading the data from the main memory into the CPU. For all other implementations the influence of the descending selectivity is higher.

Although the static array implementation is faster by a factor of five for some selectivities, we also have to consider that in absolute values, the time of iterating over a bitlist of 50 million entries (selectivity: 0.5) is between 0.08 seconds (array) and 0.26 (64-bit, hardware-supported, uncompressed). This is not bad and probably not such a dominating factor compared to the memory consumption of the different implementations shown in Figure 7.

4) Writing TIDs: In our next experiment, we analyze the time to write TIDs in the different implementation variants. This operation is done every time, when a predicate is evaluated against a column value and found to be “true”.

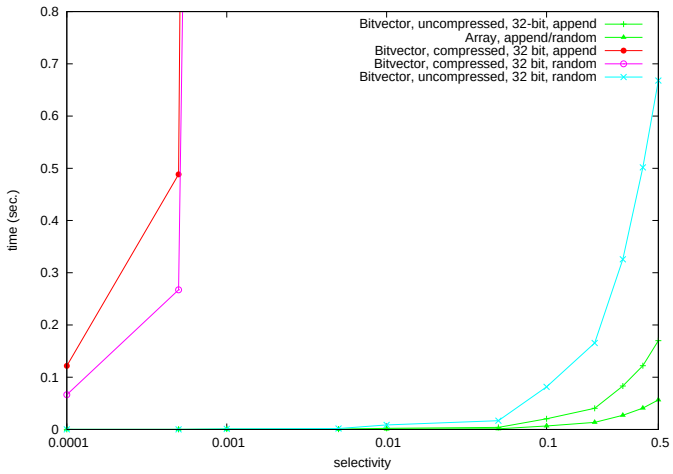


Figure 11. Measured time to write TIDs in different implementation variants for *Position Lists*

As a basic condition we can assume that writing of TIDs is mostly done in the append mode. The reason is that when evaluating a predicate on a column, this is done sequentially value by value with increasing TID values. In some complex situations, however, TIDs must be written in random order (i.e., after a previous sort operation on a column).

The results for this experiment are shown in Figure 11.

We assume 100 million datasets and measure the time to set a number of TIDs for different selectivities. So for example for a selectivity of 0.5, we have to write 50 million TIDs.

Again, the storage as an array of UINT values is the fastest solution for all selectivities. This is true for the append mode and the random order mode (from the implementation point, there is no difference between the two variants). The uncompressed bitvector turned out to be the second best solution. Based on the implementation, the solution in the append mode needs about half the time as the random write mode. This can be motivated by the fact that the number

of cache misses is lower in the append mode, than in the random mode. This characteristic increases with decreasing selectivity (0.05 and above), because the probability of the next TID being close enough to the previous TID and the corresponding memory segment (the bit) being already in the cache increases.

Compressed bitvectors behave worse. The reason for random access is that with every insertion of a TID, the compressed bitvector must be reorganized, which often has an influence up to the end of the whole compressed bitvector. This behavior occurs in the append and random modes for the WAH implementation (the WAH implementation has no special append mode, but only a *setBit(uint pos, bool value)* method to set a bit at an arbitrary position). However, the append mode could be implemented in a much more efficient way. The basic concept for the algorithm is represented in Figure 12. The idea of this implementation is that in the append mode only the last two words (LL: Last Literal, LF: Last Fill) must be considered: The last but one word, which is a literal, and the last, which is a 0-fill. Either the TID sets a bit in the last literal word, or the last fill must be split into two fills, with a literal in between (with holds the TID).

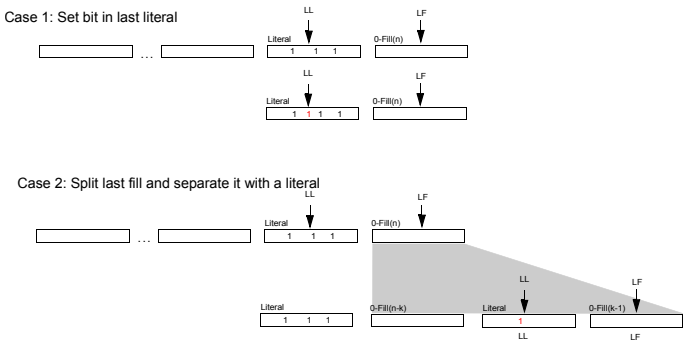


Figure 12. Appending TIDs in a compressed bitvector

To study writing of TIDs in the append mode, we extended the WAH library by the functionality described above. Figure 13 compares the time behavior for the uncompressed plain bitvector, the uncompressed WAH implementation, and our append-optimized implementation of the compressed append mode. The behavior of the append-only optimization is quite promising. It is about 25% better than using the uncompressed bitvector. The dynamic array still is the fast implementation, but, as we saw in Figure 7, memory consumption is highest, if the selectivity is low (high density values).

5) AND *operations on* Position Lists*:* Next, we perform an experiment to measure the time for *AND* operations. This is one of the basic operations performing the "WHERE" part of a query on a column store, where two or more *Position Lists* are *AND*ed (same with *OR*).

Figure 14 shows the results for the *AND* operation. As you can see, the time for *AND*'ing two uncompressed

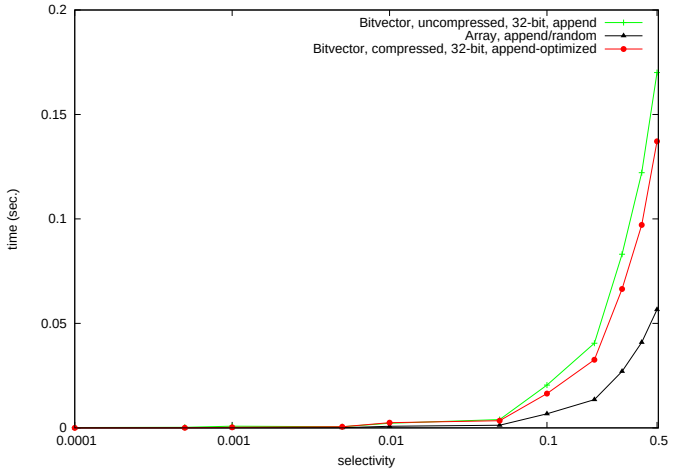


Figure 13. Comparision of appending TIDs to compressed and uncompressed bitvectors

bitvectors (both the plain bitvector implementation and the uncompressed WAH bitvector) is mostly independent of the selectivity. This can be understood, because the length of the vector is also independent of the selectivity and so the *AND* operation consists of a constant number of *and* instructions in the CPU. Comparing the uncompressed WAH bitvector and the plain bitvector, we are a little surprised. A slight overhead of the WAH implementation can be explained by the more complex algorithm and the additional memory consumption of 1/32 compared to the plain uncompressed bitvector. But our results show a significiant difference of more than 100% time penalty for the uncompressed WAH bitvector.

Also interesting are the results for the compressed bitvector and the array. While the array performs best for selectivities of 0.02 and higher, it degrades for lower selectivities (0.3 sec. for a density of 0.5). This is a little surprising, because the array implementation was one of the fastest in the previous experiments (iterating and writing TIDs). The degeneration can be explained by the caching strategies of modern CPUs. In the case of low selectivities, the two arrays grow and there is a cut-throat competition for places in the processor cache, which is why many cache misses result.

The compressed bitvector outperforms the uncompressed version for high selectivities (0.007 and above) because of its more compact representation and the ability to skip all the fill words completely. With lower selectivities, the fills get shorter and disappear later on. Hence, there is no advantage compared to the uncompressed representation. In this situation, the more complex algorithm is another drawback and leads to more instruction cache misses compared to the uncompressed version.

Again, the behavior for the array implementation is the most sensitive one. While the runtime behavior is the best for high selectivities, it completely degrades for lower se-

lectivities (density 0.03 and more).

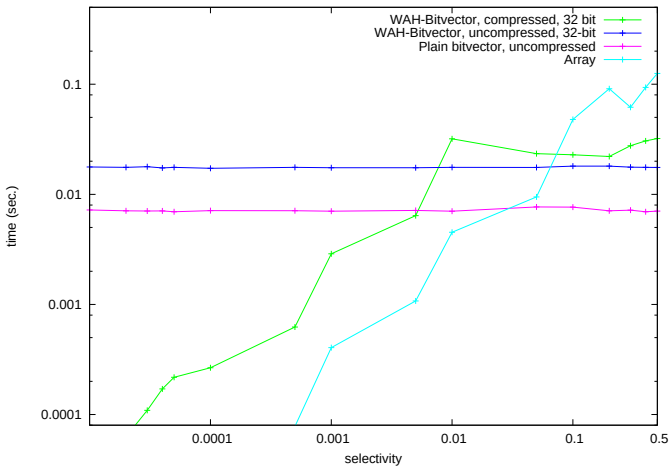


Figure 14. Measured time for *AND*ing two *Position Lists* with different implementations

6) Predicate Evaluation: In this experiment, we use *Position Lists* with different implementations (compressed bitvector, plain bitvector, array) and densities and compare their runtime behavior in evaluating the following expression:

```
column_1 = <value_1>
        and
column_2 = <value_2>
        and
column_3 = <value_3>
        and
column_4 = <value_4>
```

Each column contains 100 million datasets. In a first experiment, we formulate predicates that have the same selectivity for all four columns. In this case, the order of the evaluation of the predicates is irrelevant. We repeat this experiment with predicates of different selectivity. Figure 15 shows the execution time for the different implementations with eight different densities from 0.0001 to 0.5 (consider that the y-axis is logarithmic). The first observation is, that for densities of 1% and above the implementation of a *Position List* based on an array is about two orders of magnitude slower compared to the bitvector implementations. Also, it can be seen that for these densities, the uncompressed bitvector is the best implementation.

Additionally, we can see that the uncompressed bitvector has the same runtime behavior for all different densities. This can be explained easily by the fact that the uncompressed bitvectors have the same length for all densities and the same operations to perform. Especially for low selectivities (densities between 0.5 and 0.01), this is the preferred solution (also from the memory consumption point as shown in Figure 7). From a selectivity of 0.1%, the two other implementations perform better.

For selectivities greater than 1%, the compressed bitvector has an inferior performance compared to the uncompressed bitvector and the array implementation. This can be explained by the fact that the compression algorithm still compresses the bitvector, but only with a small compression ratio compared to the uncompressed version. The additional time results from the more complex operations while performing the *AND* operation. Starting with a density of 0.1% and lower, the compressed bitvector is the superior implementation. With a density of 0.01%, the compressed version is one order of magnitude faster than the array implementation and two orders of magnitude faster as compared to the uncompressed bitvector.

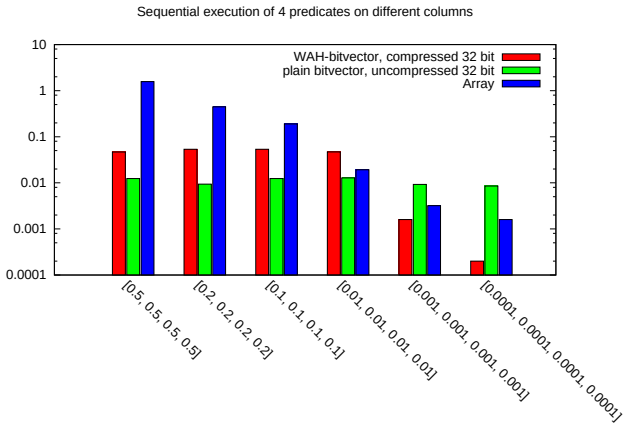


Figure 15. Measured time for the execution of 4 predicates on different columns using different implementaions for the *Position Lists*

So, as a rule of thumb, it can be said:

- If no information about the selectivity of a predicate is given, choosing an uncompressed bitvector for the implementation of the *Position List* is the best choice.
- If the density is expected to be 0.01 or higher, also use the uncompressed bitvector implementation.
- If the density is expected to be 0.01 or lower, use the compressed bitvector.

In the next experiment, we choose different selectivities for the predicates of the query. Again, we run multiple tests with different sets of selectivity. In this experiment, we vary the order in which the *Position Lists* are used to evaluate the expression. As expected, the runtime for the uncompressed bitvector is nearly independent of the density and the order of the *Position Lists*. Also expected, the runtime behavior for low selectivity queries (high density values) is bad for the array implementation (first two histogram groups). Arrays are also sensitive to the order of the *Position Lists*. The runtime behavior of the order [0.1, 0.2, 0.3, 0.4] is more

than twice as fast than the order [0.4, 0.3, 0.2, 0.1] (0.45 sec. vs. 1.0 sec.). This qualitative behavior holds for all densities. Interestingly, the behavior is inverse for densities greater than or equal to 0.1 in the case of the compressed bit vector. The reason for this behavior can be found in the implementation of the WAH algorithm. If no compression can be achieved, WAH switches automatically to the uncompressed representation. And as we have seen in Figure 14, using an uncompressed version is a little bit faster than using the compresssed version. For all lower densities, starting with the lowest density is faster. The runtime difference between the different orders degrades with lower densities (high selectivity). The last two histogram blocks also show that in the case of a first low density column, the order of the following columns is no longer critical. Nevertheless, the runtime behavior in these two last cases is determined by the low selectivity of the successive *Position Lists* and not by the first high selctivity *Position List*. If this would have been the case, the behavior of the third block (also starting with a density of 0.0001) should be expected.

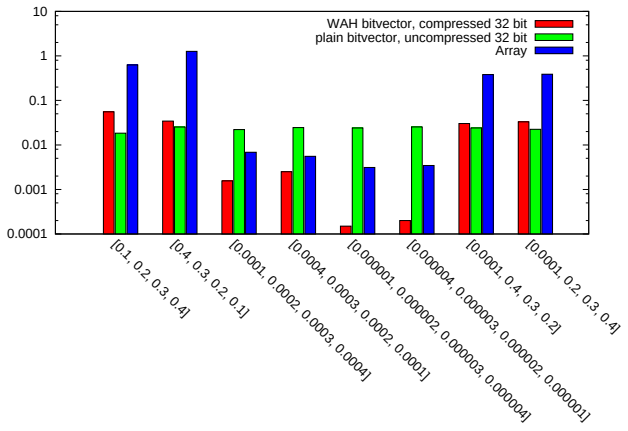


Figure 16. Measured time for the execution of 4 predicates on different columns using different implementaions for the *Position Lists*

B. Transformation between Different Representation Forms

Due to the strong influence of the selectivity, different representation forms of a *Position List* inside a complex query can be beneficial. This leads to the question how fast transformations between different representation forms can be performed. Therefore, we implement a number of transformation algorithms

($compressed \rightarrow array$, $array \rightarrow uncompressed$, $uncompressed \rightarrow array$, $array \rightarrow compressed$) and run a number of experiments, measuring the time of a *Position List* to change its internal representation. The tests are performed with different selectivities ranging from 0.0001 to 0.5. Figure 17 shows the results of these tests.

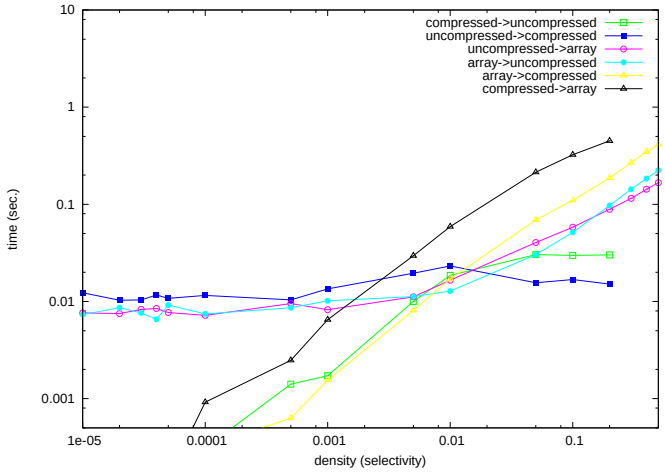


Figure 17. Time to transform different representation forms for *Position Lists*

Every line in the figure represents a concrete transformation. The x-axis represents the different densities and the y-axis the time for a transformation. Each transformation is done on a 100 million dataset *Position List*. One basic, but not surprising finding is that the transformation is faster for lower density values (high selectivity). This is based on the fact, that less TIDs must be transformed. In this case, the overall transformation time is determined by the memory footprint of the *Position List* representation, which is much smaller for the compressed representation (see Figure 7). As the transformation rules are more complex for the compressed representation, the growth in time with rising density is higher compared to the uncompressed representations (uncompressed bitvector and array). For densities greater than 0.2, WAH typically cannot compress the bitvector, because it needs at least 62 consecutive '0' values to achieve a compression. This is the reason why the transformations $compressed \rightarrow$<x> and <x>$\rightarrow compressed$ are not shown for densities greater than 0.2. The transformation to a compressed bitvector is implemented using the append-optimized algorithm from Figure 12.

Based on the results shown in Figure 14, where the array was the fastest implementation for an *AND* operation with high selectivity *Position Lists* (density lower than 0.02), but also the worst for densites greater than 0.035, we can conclude:

- For small densities (< 0.005), where the array and the compressed bitvector are the favorable implementations with respect to runtime behavior (*AND*/*OR* operations), the transformation from an uncompressed bitvector has nearly constant cost for all densities and is determined by the memory footprint of the uncompressed bitvector. Nevertheless, the transformation time is the same as when performing a single *AND* operation with two uncompressed bitvectors.

- For higher densities (> 0.01), where the cost of performing *AND* operations is up to two orders of magnitude higher when using an array implementation compared to the uncompressed bitvector, the transformation often makes sense (transformation time between 0.001 and 0.1 sec.).

VI. Conclusion and Future Work

The choice of the right data structure and algorithm for implementing *Position Lists* is not an easy task. It largely depends on the selectivity of the predicates and the operations to perform. Especially for low selectivities, the choice of the right solution is critical as was shown by the experiments.

The data structure of an array of unsigned integer values is outperformed by the uncompressed bitvector implementations by up to two orders of magnitude for low selectivities. On the other hand, it is a very good choice at high selectivities.

Uncompressed bitvectors have a predictable behavior for all selectivities, but are again outperformed by compressed bitvectors and arrays for very high selectivities.

If no information about the expected selectivity is available, using an uncompressed bitvector probably is a good choice. Depending on the selectivity and the used algorithm, the execution time is about three orders of magnitude and the uncompressed bitvector is of moderate performance.

Next, we will integrate the different implementations into our Column Store Toolkit (CSTK) [3] and perform experiments using the different implementations together with our toolkit components to measure the time behavior of our components with more complex queries like those from the TPC-H [24] benchmark.

Another interesting point is the implementation of *AND*/*OR* operations which allow *Position Lists* with different datastructures (i.e., array, compressed bitvector) as input (and output). These results can then be compared with the execution time of the version with an explicit transformation step before.

It has to be kept in mind that the ultimate goal is the development of a query optimizer for a *column store* [4].

References

[1] A. Schmidt and D. Kimmig, "Considerations about implementation variants for position lists," in *DBKDA'13: Proceedings of the Fourth International Conference on Advances in Databases, Knowledge, and Data Applications*. IARIA, 2013, pp. 108–115.

[2] P. A. Boncz, M. Zukowski, and N. Nes, "Monetdb/x100: Hyper-pipelining query execution," in *CIDR*, 2005, pp. 225–237.

[3] A. Schmidt and D. Kimmig, "Basic components for building column store-based applications," in *DBKDA'12: Procceedings of the Fourth International Conference on Advances in Databases, Knowledge, and Data Applications*. IARIA, 2012, pp. 140–146.

[4] A. Schmidt, D. Kimmig, and R. Hofmann, "A first step towards a query optimizer for column-stores," 2012, Poster presented at the Fourth International Conference on Advances in Databases, Knowledge, and Data Applications.

[5] I. Karasalo and P. Svensson, "An overview of cantor - a new system for data analysis," in *Proceedings of the Second International Workshop on Statistical Database Management, Los Altos, California*, R. Hammond and J. L. McCarthy, Eds. Lawrence Berkeley Laboratory, 1983, pp. 315–324.

[6] G. P. Copeland and S. N. Khoshafian, "A decomposition storage model," *SIGMOD Rec.*, vol. 14, no. 4, pp. 268–279, May 1985. [Online]. Available: http://doi.acm.org/10.1145/971699.318923

[7] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik, "C-store: a column-oriented dbms," in *VLDB '05: Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 553–564.

[8] P. A. Boncz, M. L. Kersten, and S. Manegold, "Breaking the memory wall in monetdb," *Commun. ACM*, vol. 51, no. 12, pp. 77–85, 2008.

[9] J. A. Khan and A. P. Shiralkar, "Article: Infobright enterprise edition analytic data warehouse technology ? an overview," *IJCA Proceedings on National Conference on Innovative Paradigms in Engineering and Technology (NCIPET 2012)*, vol. ncipet, no. 15, pp. –, March 2012, published by Foundation of Computer Science, New York, USA.

[10] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "Sap hana database: Data management for modern business applications," *SIGMOD Rec.*, vol. 40, no. 4, pp. 45–51, Jan. 2012. [Online]. Available: http://doi.acm.org/10.1145/2094114.2094126

[11] R. MacNicol and B. French, "Sybase iq multiplex - designed for analytics," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, ser. VLDB '04. VLDB Endowment, 2004, pp. 1227–1230. [Online]. Available: http://dl.acm.org/citation.cfm?id=1316689.1316798

[12] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear, "The vertica analytic database: C-store 7 years later," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1790–1801, Aug. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2367502.2367518

[13] "Oracle Database In-Memory Option — Feature — Oracle," 2013, [Online; accessed 5-May-2014]. [Online]. Available: http://www.oracle.com/us/corporate/features/database-in-memory-option/index.html

[14] "Columnstore Indexes for Fast Data Warehouse Query Processing in SQL Server 11.0," 2010, [Online; accessed 5-May-2014]. [Online]. Available: download.microsoft.com/download/8/C/1/8C1CE06B-DE2F-40D1-9C5C-3EE521C25CE9/Columnstore Indexes for Fast DW QP SQL Server 11.pdf

[15] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores vs. row-stores: How different are they really," in *SIGMOD*, 2008.

[16] N. Bruno, "Teaching an old elephant new tricks," in *CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA*. www.crdrdb.org, 2009.

[17] D. J. Abadi, D. S. Myers, D. J. Dewitt, and S. R. Madden, "Materialization strategies in a column-oriented dbms," in *Proc. of ICDE*, 2007, pp. 466–475.

[18] D. J. Abadi, S. R. Madden, and M. Ferreira, "Integrating compression and execution in column-oriented database systems," in *SIGMOD*, Chicago, IL, USA, 2006, pp. 671–682.

[19] D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, and S. Madden, "The design and implementation of modern column-oriented database systems," *Foundations and Trends in Databases*, vol. 5, no. 3, pp. 197–280, 2013.

[20] T. M. Chilimbi, B. Davidson, and J. R. Larus, "Cache-conscious structure definition," in *PLDI '99: Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation*. New York, NY, USA: ACM, 1999, pp. 13–24.

[21] S. Manegold, P. A. Boncz, and M. L. Kersten, "Optimizing database architecture for the new bottleneck: memory access," *The VLDB Journal*, vol. 9, no. 3, pp. 231–246, 2000.

[22] M. Nelson, *The Data Compression Book*. New York, NY, USA: Henry Holt and Co., Inc., 1991.

[23] K. Wu, E. J. Otoo, and A. Shoshani, "Optimizing bitmap indices with efficient compression," *ACM Trans. Database Syst.*, vol. 31, no. 1, pp. 1–38, 2006.

[24] "TPC Benchmark H Standard Specification, Revision 2.1.0," Transaction Processing Performance Council, Tech. Rep., 2002.

Instance-Based Integration of Multidimensional Data Models

Michael Mireku Kwakye[1, 2], Iluju Kiringa[1], and Herna L. Viktor[1]
[1] School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Ontario, Canada
mmire083@uottawa.ca, {kiringa, hlviktor}@eecs.uottawa.ca

[2] Faculty of Informatics
Ghana Technology University College
Accra, Greater Accra, Ghana
mmireku@gtuc.edu.gh

***Abstract*— Meta-model merging is the process of incorporating data models into an integrated, consistent model, against which accurate queries may be processed. The efficiency of such a process is very much reliant on effective semantic representation of chosen data models, as well as the mapping relationships between the schema and data instance elements of the data models. Within the data warehousing domain, the integration of data marts is often time-consuming. Intuitively forming an all-inclusive data warehouse presents tedious tasks of identifying related fact and dimension table attributes. Moreover, the ability to process queries across these disparate, but related, data marts poses an important challenge. In this paper, we introduce an approach for the integration of relational star schemas, which are instances of multidimensional data models. These instance schemas represented as data marts are integrated into a single consolidated data warehouse. Our methodology, which is based on model management operations, focuses on a formulated merge algorithm and adopts first-order Global-and-Local-As-View (GLAV) mapping models, to deliver a polynomial time, near-optimal solution of a single integrated enterprise-wide data warehouse.**

Keywords- Schema Merging, Data Integration, Model Management, Mapping Modelling Constraints, Multidimensional Merge Algorithm, Data Warehousing.

I. INTRODUCTION

The concepts of schema merging and data integration present intricate fields in databases as both have academic and industrial implications in the area of data processing. Schema merging involves integrating disparate models of related data using methods of element matching, mapping discovery, and the consolidation of data sets [2]. Schema merging adopts a concept from model management that primarily involves the integration of models and their instance schemas, and together with associated constraints. Data integration entails the consolidation of the instance data sets within the framework of a merged schema to deliver efficient query solutions [3]. The end results of schema and data integration have seen important impacts in various scientific and industrial domains. A number of the application areas are federated database systems, Enterprise Information Integration, and bioinformatics data integration. These applications continue to impact and attract attention in the need for efficient data processing and analytics.

Traditionally, most of the procedures that involve data integration have always been focused on identifying the integrating data sources, and the associated mapping correspondences of elements in the integrating data sources. Recent studies have focused instead on emphasizing the inference of semantic meaning of the elements of the data sources in integration. There usually arise various forms of problems associated with the procedural methodologies for these concepts. These challenges are the identification of prime meta-models, the expression of semantic representation of the meta-models, and the formulation of algorithms for specific meta-models and their instances. In general, these challenges make the overall procedures of data and schema integration very difficult. The conceptual processes of data integration and schema merging are derived from the fundamental operations of model management [4] [5]. Model Management operations of *match*, *compose mappings*, and *merge* offer the intuition to address the problems of data integration and schema merging within the context of multidimensional data models.

In this paper, we introduce an integration procedure for both instance schema and instance data of multidimensional data models. Our motivation is to employ the concept of model management to address the shortcomings of merge algorithm, conflict management, and technical merge requirements for integration of data marts. Our key contribution in this paper is the formulation of a novel well-defined algorithm, which is supposed to be the end-result of the overall integration process. This algorithm is capable of delivering an efficiently integrated data warehouse. Our presentation focuses on the proposition of star schema instances in our analyses. Our work, which subsumes prior work on generic models [2], draws on a number of their significant propositions made, and uses it as a background work in formalizing our intuition in a much more practical solution for merging schema and data instances of multidimensional data models. Additionally, this paper presents an extended and elaborate version to an earlier submission [1], as the assertions, methodology and evaluated results are described in further detail.

The technical contributions are summarized as follows;

- We adopt the application of a hybrid form of schema matching, in which we use both schema and data instance algorithms to deliver correct attribute mapping correspondences.
- We adopt first-order Global-and-Local-As-View mapping models in the mapping discovery procedure, which expresses the transformation of complex expressions between attributes of the instance schemas.
- We address the handling of functional dependency integrity constraints in the mapping discovery and modelling procedure.
- We identify and specify resolution measures for frequently observed conflicts that are exposed, as a result of integration of heterogeneous data marts.
- We define technical qualitative merge correctness requirements, which serve to validate the formulation of the merge algorithm.
- Most importantly, we formulate a merge algorithm that specifically deals with the integration of schema and instance data of the data marts.

The rest of the paper is organized as follows. In Section II, we review the fundamental background studies regarding data integration and schema merging. In Section III, we discuss our integration methodology, where we address the overview of the integration approach. In Section IV, we describe the adopted hybrid schema matching; and further on in Section V, we describe the mapping models discovery and modelling. In Section VI, we present the formulated merge algorithm; and address the proposition of the technical merge correctness requirements, semantics of query processing, and conflicts resolution measures in Section VII. In Section VIII, we address the implementation and evaluation of the integration methodology. We discuss the related work and comparison of other approaches in Section IX; and in Section X, we conclude, discuss open issues and the areas of future work.

II. BACKGROUND

The need of business users to access, in a timely and precise fashion, information originating from varied and heterogeneous sources of data repositories has lead to the investigation of engineered methods of efficient data integration methods and retrieval. The processes that comprise the generation of the final output of data integration largely stem from the fundamental operations of model management [4]. Models serve as data representation and as a result, different models denote different applications or domains and are modelled for different purposes.

Model management, in the field of databases, is a high-level, abstract programming language designed to efficiently manipulate schemas and mappings. It serves as the generic approach to solving problems of heterogeneity and data programmability, where concise and clear-cut mappings are manipulated to deliver desired output that supports robust operations related to certain metadata-oriented problems [4] [6] [7]. A number of these operations are; *match schemas*, *compose mappings*, *difference schemas*, *merge schemas*, *apply function*, *translate schemas* into different data models, and *generate data transformations* from mappings. The main abstractions that are needed in expressing model management operations are instance schemas and mappings. Practically, the choice of a language to express these instance schemas and mappings is vital. A model is described as a formal description of a complex application artefact, such as; database relational model, Unified Modelling Language (UML) model, or an ontology [4] [6]. A schema is an expression of a model that defines a set of possible instances, whilst and a meta-model is the language needed to express the schemas. These schemas could be relational schema, Extensible Markup Language (XML), Web Ontology Language (OWL), or Multidimensional Schema, amongst others.

Model management operations, in the form of schema matching, schema mappings, and schema merging have generally been attempted by Bernstein et al. [6], Melnik [7], and Gubanov et al. [8] to offer flexibility and efficiency in meta-data processing. To efficiently integrate different data sources, the model management operation of *match*, expressed as schema matching, serves as basis to other major operations [4]. It takes two schemas instances as input and produces a mapping between elements of the two schema instances that correspond semantically to each other [9]. Various surveys and studies have been conducted in the literature [9] [10] [11] in this direction of schema matching, of which incremental and new results have been shown to effectively deliver better solutions in arriving at precise mapping correspondences. Prior studies classify this procedure into 3 main categories. These are namely; schema-level matching, instance-level matching, and hybrid and/or composite matching. Out of these studies and surveys conducted, several concrete results have been developed to produce very high precisions. Enumerations of algorithms are Similarity Flooding (SF) [12], COMA [13], Cupid [14], SEMINT [15], iMAP [16], and the Clio Project [17] [18]. It will be noted that schema matching operations are enhanced from fields, such as; Knowledge Representation [19], Machine Learning [15] [20], and Natural Language Processing [21], where techniques are used to deliver near-automatic and semantically correct solutions.

Other forms of model management operations are *compose mappings* and *apply functions*, expressed as schema mapping discovery. These operations are normally a follow-up on the end results of a schema matching operation. Schema mapping is the fundamental operation that produces a semantic relationship between the associated elements from source and target schemas based on an earlier schema matching [17] [22] [23]. Recent studies conducted in generating schema mappings have shown that the strength of mapping relationship correspondences that exist between schema elements largely determines the degree of efficiency of the overall data integration procedure. Further works have shown that mapping correspondences modelled and expressed in terms of First-Order Logic (FOL) assertions exhibit unique characteristics, where various manipulations on mapping elements can be expressed distinctively. The authors in [24] define that an extensional mapping can be expressed as Local-As-View (LAV), Global-As-View

(GAV), Source-To-Target Tuple Generating Dependencies (s-t tgds), Second-Order Tuple Generating Dependencies (SO tgds), or other similar formalisms. More intuitively, a hybrid approach of the LAV and GAV mappings, termed as Global-and-Local-As-View (GLAV) mappings, which has been formalized to merit on the strengths of both mappings present a better mapping model for integration.

The final form of model management operation in our line of study is the *merge* operation, expressed as schema merging. Schema merging operation takes 2 meta-models and a set of mapping models, as inputs, and produces a merged meta-model, as an output, capable of representing all the elements and semantics of the input meta-models. In the generic sense, studies have been conducted and various results are addressed in the literature [2] [5] [25]. In the area of data warehousing, work done in the literature are presented in [25] [26] [27]. Additionally, the authors in [28] attempted to derive results on schema merging in relation to relational data sources, whiles merging based on semantic mappings have also been studied by the authors in [29]. A typical architecture of a merge system, as denoted in [27], is described in terms of 2 types of modules: wrappers and mediators. In terms of algorithms for merging, a generic approach was attempted in [30], whilst a proposition of an algorithm for relational sources that succeeds on a Mediated Schema Normal Form (MSNF) and Conjunctive Queries and Mappings is investigated in [28]. As part of our study, we draw on the significant propositions of generic merge in [2] and use them as background work in formalizing our algorithm in a much more practical solution for multidimensional data models.

The study of data integration and schema merging in relation to multidimensional data models has received minimal research in the literature. Cabibbo and Torlone [31] [32] [33] in their series of studies on dimension compatibility and data integration have attempted to deliver methodologies for fact and dimension tables and/or attributes integration. Riazati et al. [34] have also formalized a proposition for integration based on inferred aggregations in the hierarchies of the dimension tables, in each of the data marts. We expatiate on these approaches and perform a comparison of their work in line with our methodology in Section IX.

III. INTEGRATION METHODOLOGY

Our approach for generating a single integrated data warehouse from independent, but related, multidimensional star schemas extends from the above-mentioned concept of model management. The adopted star schema presents a modelling construct, where one large central (fact) table is referentially connected by a set of attendant (dimension) tables of varied attribute information. The fact table contains bulk data, without redundancy, whilst each dimension table contains multiple representations of attribute data instances.

We present an overview of our integration methodology, as depicted in Figure 1. The figure shows the logical and conceptual merging of the fact and dimension tables from the *Policy* and *Claims* data marts, to form an enterprise data warehouse for an Insurance industry. We explain further our motivation using Example 1 and Figure 1.

Example 1. *Suppose we have 2 data marts from an Insurance industry – Policy Transactions and Claims Transactions – and we have to integrate these data marts into an enterprise-wide data warehouse, as illustrated in Figure 1. The existence of corresponding attributes will enable the possibility of integrating the attributes of the fact and dimension tables of these data marts. A merge algorithm can be applied to the corresponding mappings to generate the integrated data warehouse needed in answering queries, as it will be posed to the integrating data marts.* ■

A. Problem Definition

In addressing our problem, we make reference to the scenario in Example 1, where we have 2 or more data marts, supposedly, in star schema models. It can be inferred that though the instance schema and data values representations in these separate data marts are different, the overlapping sets of real-world entity representations in the dimensions of the data marts seem to present a similarity. Hence, a proposition of integration for the real-world entities in each of the data marts into a single consolidated data warehouse is not improbable.

In another instance, the need to contract a merger or acquisition of companies of related business processes results in the generation of a consolidated data warehouse. This challenge in the generation of a data warehouse also falls in the paradigm of this study where each of the companies with disparate data marts or data warehouses are integrated into a single enterprise repository.

B. Overview of Integration Methodology

We outline our methodology based on 3 main streamlined procedures, namely; hybrid schema matching, mapping models discovery, and the formulation of merge algorithm. Figure 2 illustrates a description of our methodology and framework architecture in a workflow order. Here, we describe the step-wise procedures, algorithm executions, and the generated outputs. We describe in detail Hybrid Schema Matching (Procedure 1) in Section IV and Mapping Models Discovery (Procedure 2) in Section V. We also give a detailed description of the Formulated Merge Algorithm (Procedure 3) in Section VI. We address the methodology workflow in a manner where the results or output from a preceding step, e.g., Procedure 1, becomes the input for the succeeding procedure, e.g., Procedure 2. This approach ensures consistency in data processing and in the generation of the final integrated output of a data warehouse.

IV. HYBRID SCHEMA MATCHING

In our methodology, we adopt a hybrid form of schema matching, which aim to deliver efficient schema attribute correspondences. Our adoption of this hybrid approach uses the logical properties of the multidimensional schema structure in schema-based matching, and the instance data and extensions in instance-based matching, to find attribute correspondences.

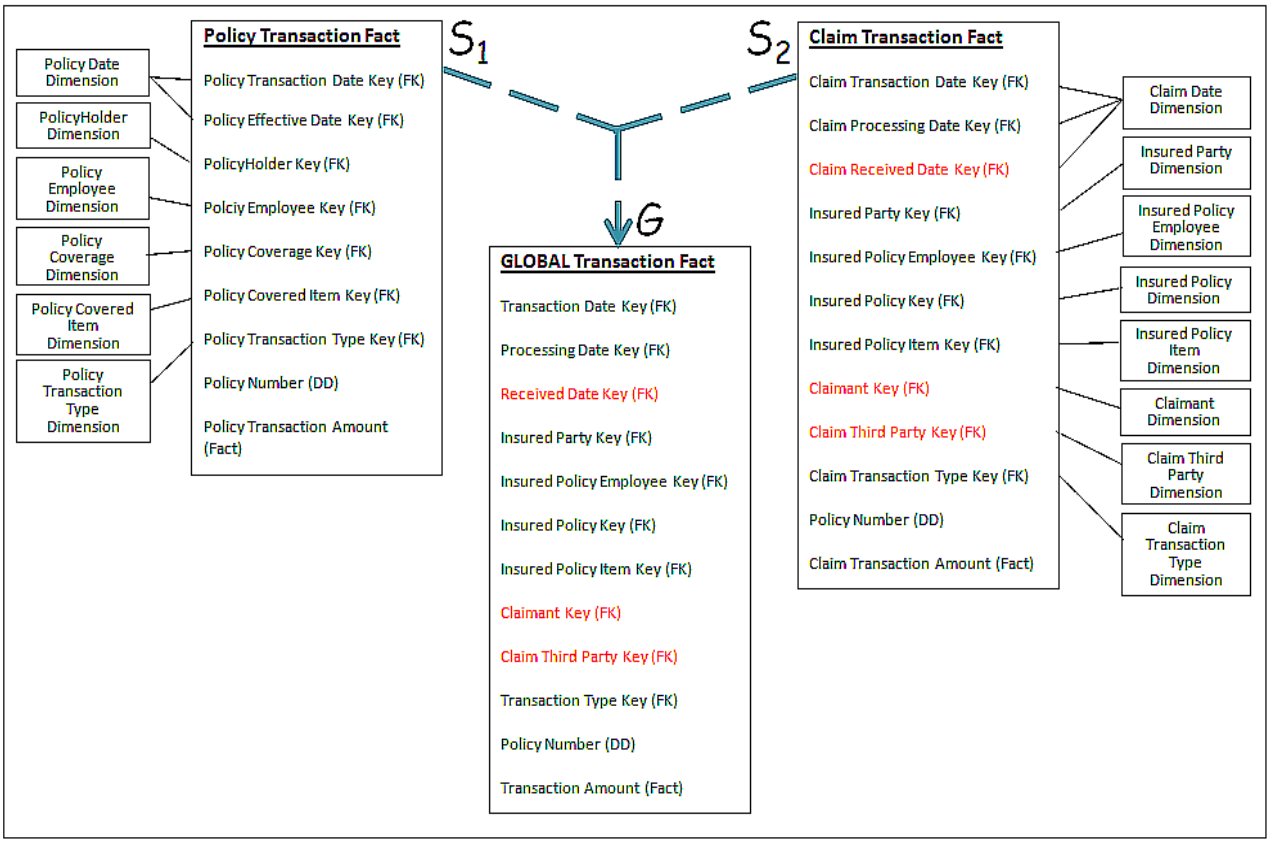


Figure 1. Logical and Conceptual Multidimensional Schema Merge

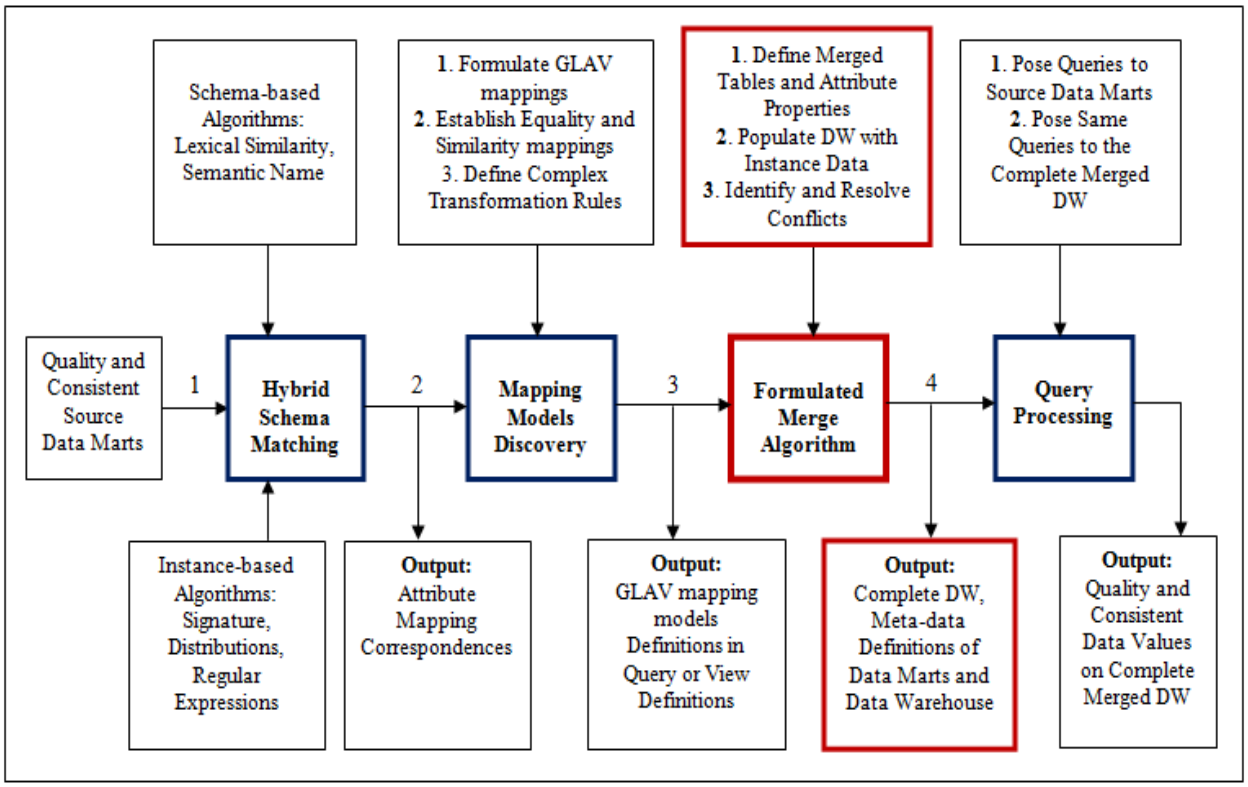


Figure 2. Workflow Framework of Integration Methodology

A. Schema-based Matching

We adopted schema-based algorithmic techniques in the form of Lexical Similarity and Semantic Names.

The Lexical Similarity is an algorithm technique based on the linguistic form of schema matching, in which string *names* and *text* are used to semantically find similar schema elements. This algorithmic technique defines a measure of the degree to which the word sets of 2 given strings are similar, and discovers maximum weight subsequence of the strings that are common to each other. The algorithm determines similarity based on schema string names and text, equality of names, equality of synonyms, homonyms, abbreviations, and similarity of common substrings, amongst others [35].

The Semantic Names, on the other hand, is an algorithmic technique based on the semantic deduction of the schemas and their characteristics. The algorithmic technique is reliant on the schema structure and the properties of the elements, and enforces on varied forms of constraints. It uses criteria such as, type similarity and metadata in relation to table name, attribute names, schema data types, value ranges, precision, uniqueness, optionality, relationship types, cardinalities, key properties, referential constraints, amongst others, to match attributes [35].

We use Example 2 to illustrate the schema-based form of finding mapping correspondences.

Example 2. *Following up on Example 1, suppose we want to merge the dimensions of* ***DimPolicyHolder*** *and* ***DimInsuredParty*** *from Policy and Claims data marts, respectively. The application of Lexical Similarity algorithm will produce mapping correspondences, such as:*

1. $DimInsuredParty.InsuredPartyKey \approx DimPolicyHolder.PolicyHolderKey$

2. $DimInsuredParty.FullName \approx DimPolicyHolder.FamilyName, DimPolicyHolder.GivenName, DimPolicyHolder.CityName, DimPolicyHolder.DistrictName$

3. $DimInsuredParty.StreetAddress, DimInsuredParty.EmailAddress \approx DimPolicyHolder.Address$

Moreover, the application of the Semantic Names algorithm will offer an improved schema matching using the data types, relationships types and constraints, and value ranges. This algorithmic matching enforces on the already generated correspondence in the Matching (1), where ***Int*** *data types and* ***Primary Key*** *constraints for both attributes of* ***DimInsuredParty.InsuredPartyKey*** *and* ***DimPolicyHolder.PolicyHolderKey*** *are used for element relationship mapping. For Matching (2), the* ***DimPolicyHolder.CityName [varchar(18)]*** *and* ***DimPolicyHolder.DistrictName [varchar(15)]*** *attributes were eliminated to deliver mapping correspondence, as in:*

2. $DimInsuredParty.FullName\ [varchar(60)] \approx DimPolicyHolder.FamilyName\ [varchar(25)], DimPolicyHolder.GivenNames\ [varchar(40)]$

The above mapping correspondence is generated as a result of the semantic representations of data type and precision, such as ***varchar(60)*** *for* ***DimInsuredParty.FullName*** *to correspondingly infer on* ***varchar(25)***, ***varchar(40)*** *for both* ***DimPolicyHolder.FamilyName*** *and* ***DimPolicyHolder.GivenName****, respectively.* ■

B. Instance-based Matching

The instance-based algorithms that were adopted are Signature, Distributions, and Regular Expressions. These algorithmic techniques are based on the instance data contained in the schemas and infer on the characteristics, meaning and similarity in the data values, as well as the relationship to other data set contained in the schema. The Signature algorithm uses the similarity in the actual data values contained in the schemas and their signature based on data sampling. The technique uses sampled data to find relationships where a weighting value is assigned to certain classes of words in the data [35]. This sampling of data is based on the valid values of sampling size and also the rate of the sampling. The determination of match signature is done by clustering according to their distance measure, either by Euclidean distance [36] or Manhattan distance [37].

The Distributions algorithm discovers mapping correspondences based on the common values in the instance data contained in the schemas. The algorithm also uses data sampling to aid the discovery function to find relationships between attribute data values, where the frequent occurrence of most data values for a particular attribute in relation to another attribute determines the candidacy of matching correspondence. Prior attempts of methodologies within the domain of machine learning that aid in the discovery of correspondences are A-priori and Laplacian [38].

The Regular Expressions algorithm uses textual or string searches based on regular string expressions or pattern matching. A simple regular expression will be an exact character match of attribute data values or of the common substrings contained in the instance data. This algorithm also uses data sampling to aid the discovery function of finding relationships between attribute data values [39].

We use Example 3 to illustrate a generalized form of instance-based algorithm.

Example 3. *Following up on Example 2, we complement the results of the initial schema-based mapping correspondences with a generalized instance-based mapping to produce a final semantically correct mapping correspondence for the Matching (3), as in:*

3. $DimInsuredParty.StreetAddress \approx DimPolicyHolder.Address$

This final matching was attained because of the data values and extensions from the dimension attributes. A representation of the instance data values contained in ***DimPolicyHolder.Address*** *are {39 Baywood Drive, 178 Flora Ave., 79 Golden Rain St.}, where as data values contained in* ***DimInsuredParty.StreetAddress*** *and* ***DimInsuredParty.EmailAddress*** *are {40 Roslyn St., 68 Hastings Drive, 48 Whitehall Avenue} and*

{amartens@cybserv.com, drice@vipe2k.com, jtausig@fitexes.com}, respectively. ■

In general, the output generated from this step of Procedure 1, is a set of mapping correspondences between the elements of the instance schema structure and instance data values of the heterogeneous data sources.

V. MAPPING MODELS DISCOVERY

In the second procedure of our integration methodology, we discuss the adoption of first-order GLAV mapping models. We also discuss the merits of the mapping model, whilst highlighting the suitability for our integration approach. Moreover, we discuss the handling of possible functional dependency integrity constraints as they occur in the mapping models. This step utilizes the output of Procedure 1, as inputs, to aid in the discovery and establishment of mapping models.

Definition 1. (First-Order Mapping): Let $\mathcal{M} = (S, T, f)$ represent a mapping model from Source, S and Target, T schemas. Let $a \in \{S \cup T\}$ represent disjoint variable element where a denotes $\{a_1, a_2, \dots, a_n\}$. The mapping assertion, $\mathcal{M}$ is said to be in first-order if $f: \{\forall a \, (S(a) \rightarrow T(a))\}$, where f represents the logical view from the Source to the Target. ■

We adopt a first-order GLAV mapping model formalism [40]. This mapping formalism is based on first-order logic assertions, where elements are finitely mapped using the functional relations existing between them. Our motivation is founded on the expressiveness of the correspondences that exist between the attributes of the schemas [3]. The GLAV mapping model combines mapping formalisms from both the Local-As-View (LAV) and Global-As-View (GAV) mappings [40]. This mapping model expresses mapping views where the extensions of the source schemas provide subsets of tuples satisfying the corresponding view over the global mediated schema. Moreover, an equivalent number of attribute view definitions are expressed in both the LAV and GAV queries [3]. One other unique feature of the GLAV mapping modelling is the expression of multi-cardinality mappings between mapping elements. This enables the expression of complex transformation formulas, which is much useful in our integration methodology [24].

Definition 2. (Equality Mapping): Let $\mathcal{M} = (S, T, f)$ represent a mapping for Source, S and Target, T schemas. The assertion $f: \{\forall x \forall y \, (S(x, y) \rightarrow \exists z \, T(x, z))\}$ for disjoint variable elements x, y, z is an Equality mapping, such that $y = \text{z}$. ■

Definition 3. (Similarity Mapping): Let $\mathcal{M} = (S, T, f)$ represent a mapping for Source, S and Target, T schemas. For disjoint element variables x, y, z the assertion $f: \{\forall x \forall y \, (S(x, y) \rightarrow \exists z \, T(x, z))\}$ is a similarity mapping, such that $g(y) = z$ where g denotes or encloses a complex transformation expression. ■

In this procedural step, 2 forms of mapping relationships were adopted, namely; *Equality* and *Similarity* mapping relationships. An equality mapping represents a one-to-one mapping, whilst a similarity mapping also represents a one-to-many or many-to-many mapping. The defined classifications were based on expressive characterization of relationship cardinality, and the attribute semantic representation, amongst others [39]. We used these forms of mapping relationships in a GLAV mapping model, as explained in Example 4.

Example 4. *Using the scenario described in Example 1, suppose we want to integrate the* ***DimPolicyHolder*** *and* ***DimInsuredParty*** *dimensions from Policy and Claims data marts, respectively, into* ***DimInsuredPolicyHolder*** *dimension. The Datalog queries for the GLAV mapping model will be expressed as:*

DimInsuredPolicyHolder *(InsuredPolicyHolderKey, InsuredPolicyHolderID, InsuredPolicyHolderName, BirthDate, ProvinceState, Region, City, Status):-*
DimPolicyHolder *(PolicyHolderKey, PolicyHolderID, PolicyHolderFamilyName, PolicyHolderGivenName, DateOfBirth, ProvinceState, CityName, Status),*
DimInsuredParty *(InsuredPartyKey, InsuredPartyID, InsuredPartyFullName, BirthDate, Province, Region, City)*

In this Datalog query, the existence of corresponding attributes in both dimensions automatically expresses an equality representation in the merged dimension. Additionally, a similarity relationship is established where, for example, ***DimPolicyHolder.InsuredFamilyName*** *and* ***DimPolicyHolder.InsuredGivenName*** *attributes are mapped onto the merge attribute of* ***DimInsuredPolicyHolder.InsuredPolicyHolderName****. Moreover, local attributes of* ***DimPolicyHolder.Status*** *and* ***DimInsuredParty.Region*** *from Policy and Claims data marts, respectively, are also included in the merged dimension schema instance.* ■

A. Propositions for GLAV Mapping Models

We further summarize a number of the characteristic features that merit the choice of the GLAV mapping model. This mapping model represents a suitable form of manipulation of the mapping relationships that exists between the instance schema attributes, as well as the instance data values, contained in the star schema data marts. Moreover, the GLAV mapping features offer the relationship needed for the generic application of the merge algorithm for disparate and heterogeneous schema and data instances.

Automatic Mapping Generation. It is a mapping formalism that facilitates the (semi-)automatic generation of schema mappings from heterogeneous instance schemas. This is evident in cases where mapping correspondences are incomplete or incorrect. This characteristic feature also offers the ability to incrementally modify mappings as correspondences change.

Mapping Reusability. The mapping model facilitates the composition of sequential mappings that enables the re-use of mappings when the instance schemas are different or change. This functionality offers the capability to

reformulate queries against one schema into queries on another schema during data integration.

Data Translation & Exchange. The semantics of such a mapping and its data exchange capabilities offers a data translation from one schema to another. Moreover, the mapping offers the transformation from one representation to the other during data exchange based on specifications.

Runtime Functionality. The mapping formalism expresses the capabilities for runtime executables; for example, to generate view definitions, query answering, and generation of XSLT transformations, amongst others.

Data Manipulation. The semantics of the GLAV mapping model makes it easily applicable and manipulated by mapping tools; for example, the IBM InfoSphere Data Architect [41], Microsoft BizTalk Mapper [42], amongst others.

Query Code Generation. The mapping formalism offers a platform where query codes are generated based on the mapping relationships. This is evident where efficient queries or transformations in various languages (e.g., native SQL) can implement the formulated mappings.

B. Functional Dependency Mapping Integrity Constraints

In our methodology, the adopted first-order GLAV mapping modelling can be enhanced to deliver efficient relationships between the attributes of the schema instances, by the application of mapping integrity constraints. One form of mapping integrity constraint is *Functional Dependencies* of the dimension instance schema attributes.

Definition 4. (Functional Dependency): Suppose $\mathcal{D} = (\mathcal{A}_1, \dots, \mathcal{A}_n)$, for $n \geq 2$ attributes represent a dimension instance schema. The assertion of functional dependency, $\mathcal{FD}$ constraint stipulates that a set of attributes $\{\mathcal{A}_1, \dots, \mathcal{A}_u\} \in \mathcal{D}$ uniquely determines another set of attributes $\{\mathcal{A}_{u+1}, \dots, \mathcal{A}_n\} \in \mathcal{D}$, based on a key constraint, say $\mathcal{A}_1$, such that $\mathcal{FD}$: $\{\forall\mathcal{A}_1\forall\mathcal{A}_u\forall\mathcal{A}_{u+1}(\mathcal{D}(\mathcal{A}_1, \mathcal{A}_u) \wedge \mathcal{D}(\mathcal{A}_1, \mathcal{A}_{u+1}) \rightarrow (\mathcal{A}_u = \mathcal{A}_{u+1}))\}$. ■

From the definition above, it can be inferred that the set of attributes $\{\mathcal{A}_1, \dots, \mathcal{A}_u\} \rightarrow \{\mathcal{A}_{u+1}, \dots, \mathcal{A}_n\}$ uniquely, where the data instance tuple values in attribute set, $\{\mathcal{A}_{u+1}, \dots, \mathcal{A}_n\}$ are dependent on, or can be derived from the tuple values in attribute set, $\{\mathcal{A}_1, \dots, \mathcal{A}_u\}$.

We use Example 5 below to illustrate the occurrence of functional dependency integrity constraint, as part of the mapping discovery and modelling.

Example 5. *Suppose an integrity constraint exists on the instance schema dimension* ***DimPolicyCoveredItem*** *in the Policy data mart where the attribute* ***DimPolicyCoveredItem.PolicyCoveredItemID*** *functionally determines the set of attributes* ***DimPolicyCoveredItem.PolicyCoveredItemType*** *and* ***DimPolicyCoveredItem.CoveringPeriod****. Suppose a principal mapping correspondence is established between the Natural Key attributes of* ***DimPolicyCoveredItem.PolicyCoveredItemID*** *and* ***DimInsuredPolicyItem.InsuredPolicyItemID.*** *Moreover, suppose mapping correspondences are established between attributes* ***DimPolicyCoveredItem.PolicyCoveredItemType****,* ***DimPolicyCoveredItem.CoveringPeriod*** *and* ***DimInsuredPolicyItem.ItemForm,*** ***DimInsuredPolicyItem.PolicyPeriod,*** *respectively. The modelling of first-order GLAV mappings between the dimensions* ***DimPolicyCoveredItem*** *and the* ***DimInsuredPolicyItem*** *will result in an automatic instance functional dependency constraint in* ***DimInsuredPolicyItem*** *dimension. This dependency is expressed in the set of attributes* ***DimInsuredPolicyItem.ItemForm*** *and* ***DimInsuredPolicyItem.PolicyPeriod,*** *to functionally depend on the* ***DimInsuredPolicyItem.InsuredPolicyItemID*** *attribute. This dependency association is modelled in the merged table and its attributes for each of the integrating table schema instances.* ■

It will be affirmed that the dependency association between attributes complements the derivation of the merge schema and data instances. Moreover, the dependency constraint enables the population of data instance tuple values, especially in the Steps (10) and (11) in the merge algorithm (Algorithm 1) in Section VI.A. This can be addressed in the scenario where, if the tuple values for the set of $\mathcal{A}_u$ attributes are known, say a_u, then the tuple values for the set of $\mathcal{A}_{u+1}$ attributes, say $a_{u+1,}$corresponding to and depending on a_u can be determined by looking them up in tuple values of the a_u.

The output generated from Procedure 2 step, is a set of mapping models outlining the types of *Equality* and *Similarity* mapping relationships. The output expresses the merge schema definitions, schema constraints, and complex transformations for the one-to-many and/or many-to-many relationships of the heterogeneous data source elements.

VI. FORMULATED MERGE ALGORITHM

We present and describe an elaborate merge algorithm (Algorithm 1) for integrating the instance schema and data of data marts fact and dimension tables. We further provide a summary of the algorithm, and conclude the section by presenting a computational complexity analysis of the formulated algorithm.

A. Merge Algorithm

The merge algorithm (Algorithm 1) is formulated to generate the single consolidated data warehouse from different related data marts, modelled as star schemas instances.

B. Merge Algorithm Summary

The merge algorithm primarily performs 2 levels of integration.

Firstly, the integration of the instance schema structure, which comprises the attribute relationships and properties for the fact and dimension tables. These procedures are described in Steps (1) to (9). Steps (1) to (4) initialize and generate the integrated schema tables. Steps (5) to (7) describe the generation of attributes for the integrated tables.

Algorithm 1: Multidimensional Instance Schema and Data Integration

Input:
(a) A set of *star schema* data marts, *A* and *B*
(b) A set of *first-order* GLAV mapping model; $Mapping_{AB}$, consisting of $factMapping_{AB}$ and $dimMapping_{AB}$
(c) An optional designation of a data mart, *A* or *B*, as the $preferredDataMart$;

Output:
(a) A single consolidated star schema instance data warehouse free of *duplicate* and *redundant* schema and instance data.
(b) A metadata consisting of data definition of the integrating data marts and the single consolidated data warehouse.

Procedure:

Initialization
(1) ***Let*** $mergeDataWarehouse \leftarrow NULL$

Generate Merged Table
(2) ***For each*** $correspondingMappingType \in factMapping_{AB}$ ***do***
  (a) ***If*** $corrrespondingMappingType = NULL$ ***then***
    i. ***Return*** $mergeDataWarehouse \leftarrow NULL$
  (b) ***Else***
    i. ***Let*** $mergeDataWarehouse \leftarrow mergeFactTable \in \{factTableA, factTableB\}$
(3) ***Repeat*** Step (2) for each $mergeDimTable$ using $dimMapping_{AB}$, add $\{nonCorrespondingDimTable\}$
(4) ***Return*** $mergeDataWarehouse \supset \{mergeFactTable, \{mergeDimTable, nonCorrespondingDimTable\}\}$

Merged Table Attribute Relationships
(5) ***For each*** $correspondingMappingType \in factMapping_{AB}$ ***do***
  (a) ***Let*** $mergeFactTable \leftarrow NULL$
  (b) ***If*** $correspondingMappingType =$ "*Equality*" ***then***
    i. ***Let*** $mergeFactAttribute \leftarrow definedAttribute \in \{factMapping_{AB} \in preferredDataMart\}$
  (c) ***Else If*** $correspondingMappingType =$ "*Similarity*" ***then***
    i. ***Let*** $mergeFactAttribute \leftarrow definedAttribute \in factMapping_{AB}$
(6) ***For each*** $nonCorrespondingAttribute \in \{factTableA, factTableB\}$ ***do***
  (a) ***If*** $nonCorrespondingAttribute \notin \{mergeAttribute\}$ ***then***
    i. ***Let*** $mergeFactAttribute \leftarrow nonCorrespondingAttriubte$
  (b) ***Return*** $mergeFactTable \supset \{mergeFactAttribute, nonCorrespondingAttribute\}$
(7) ***For each*** $correspondingMappingType \in dimMapping_{AB}$ ***do***
  (a) ***Repeat*** Step (3) for each $correspondingAttribute \in \{dimTableA, dimTableB\}$
  (b) ***Repeat*** Step (4) for each $nonCorrespondingAttribute \in \{dimTableA, dimTableB\}$
  (c) ***Return*** $mergeDimTable \supset \{mergeDimAttribute, nonCorrespondingAttribute\}$

Merged Table Attribute Properties
(8) ***For each*** $mergedFactAttribute \in mergeFactTable$ ***do***
  (a) ***Let*** $mergeAttributeTypeValue \leftarrow definedAttributeType \in factMapping_{AB}$
(9) ***Repeat*** Step (6) for each $mergeDimTable$ using $dimMapping_{AB}$

Dimension Tables Data Population
(10) ***For each*** $mergeDimTable$ ***do***
  (a) ***If*** ($keyIdentifierConflict$ OR $multipleEntityRepresentation$) = $TRUE$ ***then***
    i. ***Let*** $entityKeyIdentifier \leftarrow surrogateKey \in preferredDataMart$
  (b) ***Else***
    i. ***Let*** $entityKeyIdentifier \leftarrow (newSurrogateKey \equiv primaryKey) \in nonPreferredDataMart$

Fact Table Data Population
(11) ***For each*** $mergeFactTable$ ***do***
  (a) Load fact records using $entityKeyIdentifier \in \{surrogateKey, newSurrogateKey\}$
(12) ***Let*** $mergeDataWarehouse \supset \{mergeFactTable, \{mergeDimTable, nonCorrespondingDimTable\}\}$
(13) ***Return*** $mergeDataWarehouse$

Finally, Steps (8) and (9) describe the derivation of attribute property values of the merged fact and dimension tables.

Secondly, the algorithm performs integration of the instance data values contained in the star schema data marts. This involves the population of these instance data from the data marts fact and dimension tables into the merged tables in the data warehouse. Steps (10) to (13) describe these procedures of data population.

C. Merge Algorithm Computational Complexity

The merge algorithm presented in previous section, Sub-section A, is projected to run with a low worst-case complexity of a polynomial time.

The Initialization step in Step (1) requires a complexity of $O(n)$, whiles the Step (2) takes $O(n^2 \log m)$ to derive a merged fact table and dimension tables, for n number of tables and m number of corresponding attributes.

The iterative processes of Step (5) and Step (6) involves a computation running time of $O(k + n^2 \log m)$ to generate the table attributes and their relationships, for n number of tables, m number of corresponding attributes, and k number of non-corresponding attributes.

Finally, the steps from Step (8) to Step (12) require running time of $O(k + m)$ for the iterations performed. An overall worst-case complexity of $O(n) + O(k + m) + O(k + n^2 \log m)$ is attained in running the merge algorithm to generate the single consolidated data warehouse.

We also give a detailed Proof of Correctness of the merge algorithm in Appendix XI.

VII. PROPOSITIONS OF MULTIDIMENSIONAL INSTANCE SCHEMA AND DATA INTEGRATION

In this section, we propose technical qualitative requirements necessary for producing an efficient single consolidated data warehouse. We also describe the semantics of query processing on integrated instances of multidimensional data models. We finally propose and describe the resolution of identifiable conflicts associated with the integration of the data marts.

A. Merge Correctness Requirements

The single consolidated data warehouse that is generated as a result of the implementation of the merge algorithm needs to satisfy proposed requirements, to ensure the correctness of the data values from the queries that would be posed to it.

Drawing on the propositions in the requirements defined by the authors in [1] for merging generic meta-models, we performed a gap analysis and extend on their propositions in relation to generating a data warehouse. Hence, we formulate and describe a set of correctness requirements in relation to merging of multidimensional star schemas. We outline the set of *Merge Correctness Requirements* (MCR) that validates the formulated merge algorithm needed for the generation of a single consolidated data warehouse.

Dimensionality Preservation. For each kind of dimension table connected to any of the integrating fact tables, there is a representation of corresponding dimension also connected to the merged fact table.

Measure and Attribute Entity Preservation. All fact or measure attribute values in either of the integrating fact tables are represented in the merged fact table. Additionally, attributes in each of the dimension tables are represented through an equality or similarity mapping. Finally, an automatic inclusion for non-corresponding attributes in the merged tables, based on the condition of non-attribute redundancy or duplication, is satisfied.

Slowly Changing Dimension Preservation (SCD). SCD is the occurrence where an entity in a dimension exhibits multiple instance representations, based on the varied changes in instance data values for the key dimensional attributes, over a time period [43]. For such dimensional entity occurrences, the merged dimension should offer an inclusion of all the instance data representations from each integrating dimension. Hence, an automatic inclusion of attributes that contribute to the dimensional change in the merge dimension is satisfied.

Attribute Property Value Preservation. The merged attribute should preserve the value properties of the integrating attributes, whether the mapping correspondence is an equality or similarity mapping. Equality mapping should be trivially satisfied by the *UNION* property for all attributes. For a similarity mapping, the transformation expression should have the properties to be able to satisfy the attribute property value of each integrating attribute.

Definition 5. (Surrogate Key): Let $\mathcal{D}_i$ represent a dimension table for a multidimensional model, $\mathcal{B}$ such that $\mathcal{D}_i \in \{\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_n\}$ for $i \leq n$. Let $\mathcal{E}$ represent each entity of a dimension, $\mathcal{D}_i$ such that $\mathcal{E} \in \mathcal{D}_i$. The identifier, $\mathcal{K}$ is said to be a Surrogate Key for $\mathcal{E}$ such that $\mathcal{K}_m \equiv \mathcal{E}_m$ ■

Tuple Containment Preservation. A Surrogate Key is the dimensional attribute that uniquely identifies each instance data value tuple of an entity representation. The single consolidated data warehouse should offer the containment of all unique tuples from the data marts for correctness in query answering. This ensures the preservation of all *Surrogate Keys* needed in identifying each dimensional entity.

B. Merge Algorithm Technical Requirements Summary

The integration methodology adopted by the authors largely satisfies the technical requirements, as a proposition for merging disparate data marts. We summarize the merge algorithm (Algorithm 1) in fulfilment of the technical *Merge Correctness Requirements* (MCRs) outlined in Section VI.A.

a) Step (2) satisfies Dimensionality Preservation: Each fact and dimension table is iterated to form the Merged Fact Table.

b) Steps (3), (4), (5) satisfy Measure and Attribute Entity Preservation: All the attributes contained in the Fact or Dimension Tables are represented in the Merged Table (Fact or Dimension) through equality or similarity mapping.

c) Steps (6) and (7) satisfy Attribute Property Value Preservation: Value properties of attributes are represented for each of the Fact or Dimension Tables.

d) Step (8) satisfies Slowly Changing Dimension Preservation and Tuple Containment Preservation: Entity representations from the different data marts are included in the merged dimensions.

e) Steps (9), (10) satisfy Tuple Containment Preservation: Tuple data values from each of the data marts are populated in the merged data warehouse.

C. Semantics of Query Processing on Integrated Instances of Multidimensional Data Models

The type of queries that are processed on multidimensional data models are based on Online-Analytical Processing (OLAP). There are a few problems that are inherent with OLAP query processing, and these are addressed as follows. On one hand, is the problem of incomplete data that arise from missing data values, and imprecise data values of varying extent. In our approach, the possibility of having missing data values, in relation to non-corresponding merge attribute, from the star schemas is highly probable. Moreover, the varying granularities caused by the different degrees of precision in the data values from the combined instance data of different star schemas, exposes a non-uniform representation of the data values needed for analytical reporting.

Definition 6. (Dimension Hierarchy): A hierarchy, $\mathcal{H}$ comprising a dimension, $\mathcal{D}$, is a 2-tuple $(\mathcal{L}_n, \nearrow)$ where $\mathcal{L}_n$ is a collection of levels and each $\mathcal{L}_i \in \{\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_n\}$, $i \leq n$, and $\nearrow$ is a parent-child relation of two levels, say $\mathcal{L}_i$ and $\mathcal{L}_j$, such that a data instance element in $\mathcal{L}_i$ rolls up to a data instance element in $\mathcal{L}_j$, denoted by $(\mathcal{L}_i \nearrow \mathcal{L}_j)$. This roll-up relationship forms a partial order over the levels. ■

Definition 7. (Strict Hierarchy): For a dimension schema instance $\mathcal{D}$, any hierarchy $\mathcal{H} \in \mathcal{D}$, is said to be strict if for every pair of levels $\mathcal{L}_i, \mathcal{L}_\mathrm{j}$ with the partial ordering $(\mathcal{L}_i \nearrow * \mathcal{L}_j)$, which are through different paths, say $\{\mathcal{L}_\mathrm{i}, \mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_\mathrm{u}, \mathcal{L}_j\}$ and $\{\mathcal{L}_\mathrm{i}, \mathcal{L}_\mathrm{a}, \mathcal{L}_\mathrm{b}, \dots, \mathcal{L}_\mathrm{v}, \mathcal{L}_j\}$, and for every instance data element e in $\mathcal{L}_i$, there exist a roll-up function composition that holds for the condition: $\int_{\mathcal{L}_i}^{\mathcal{L}_1} \circ \int_{\mathcal{L}_1}^{\mathcal{L}_2} \circ \dots \circ \int_{\mathcal{L}_u}^{\mathcal{L}_j}(e) = \int_{\mathcal{L}_i}^{\mathcal{L}_a} \circ \int_{\mathcal{L}_a}^{\mathcal{L}_b} \circ \dots \circ \int_{\mathcal{L}_v}^{\mathcal{L}_j}(e)$ ■

On the other hand, the problem of imperfections inherent in the hierarchy levels of dimensional tables also places an overhead impact on query processing for multidimensional data models. Hierarchies enable drill-down and roll-up in the aggregate data, and as a result, multiple hierarchies in a particular dimensional entity support different aggregation paths within the dimension. Different forms of strict and non-strict hierarchies are exhibited in the dimensional entities of multidimensional data models. Strict hierarchies exhibit a phenomenon where a dimension data instance item or child level element has only one parent level element enforcing a constraint restriction on the data values that are rolled-up during aggregation. Hence, strictness in hierarchies ensures a consistency in the instance data values that are used in roll-up functions. Non-strict hierarchies exhibit a phenomenon where a dimension data instance item or child level element has several elements at the parent levels, thus allowing flexibility in the data aggregation.

Pedersen et al. [44] proposed requirements that a multidimensional data model should satisfy in order to fully support OLAP queries. These are outlined as; explicit hierarchies in dimensions, multiple hierarchies, non-strict hierarchies, handling different levels of granularity, and handling imprecision amongst others. These requirements give insights into how OLAP tools manage the raw data values and how data values are expressed during analytics.

As part of our study, query processing is handled in relation to the proposition in [44]. The adopted star schema model offers a platform for basic SQL star-join optimization during the processing of data values for analytical representation. The ability of structured cube modelling for each of the dimension elements by OLAP representations offers the medium for the individual hierarchies in the dimensional entities to be captured explicitly. The hierarchies and their data manipulations are captured using either, grouping relations and functions, dimension merging functions, roll-up functions, level lattices, hierarchy schemas and instances, or an explicit tree-structured hierarchy as part of the cube.

Different forms of aggregations are computed in the approach of query processing on the generated data warehouses. These aggregations are made possible because of the defined hierarchies established in the dimensional entities. The aggregations are represented in functions such as addition computations, average calculations, and constant functions through an OLAP operation of summarizability.

Definition 8. (Summarizability): A hierarchy $\mathcal{H}$ is summarizable if for all levels $\mathcal{L}_i \in \{\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_n\}$, for $1 \leq i \leq n$, of this hierarchy, the single-level aggregated measure m with $\mathcal{L}_i$ granularity, can be computed by summing up data instance tuple values of a single-level specified for measure m for any $\mathcal{L}_k \in \{\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_n\}$, for $1 \leq k \leq n$, granularity appearing along a path from the bottom level to $\mathcal{L}_i$. ■

Summarizability is a conceptual property of multidimensional data models where individual aggregate results can be combined directly to produce new aggregate results. In a summarizable hierarchy, the aggregated values for a measure at a level granularity can be obtained by aggregating the elements of any level of hierarchy, which directly or indirectly rolls up to the desired level. This characteristic feature guarantees the correctness of aggregated values in the resultant data warehouse.

D. Conflicts Identification and Resolution

The integration of meta-data models is generally coupled with different forms of conflicts in either the instance schema or instance data. These conflicts are resolved through different propositions from the formulated algorithm, and based on the semantic representation of the meta-data models and their instance schemas. In our integration approach, we identify and propose resolution measures for likely to occur conflicts, which are frequently encountered during merging.

Identifier Conflicts. These conflicts arise as a result of the same identifier for different real-world entities in the merged dimension. These categories of conflicts are practically exposed as a result of the possibility of different entities from the integrating data marts having the same surrogate key identifier in their individual dimensions. A resolution measure for these conflicts is explained in Example 5.

Example 6. *Suppose we aim to merge the employee dimensions into a single merged dimension, using* ***DimPolicyEmployee*** *and* ***DimInsuredPolicyEmployee*** *from Policy and Claims data marts, respectively. In such an integration procedure, it happens that an instance data value, Employee P from* ***DimPolicyEmployee*** *and an instance data value, Employee Q from* ***DimInsuredPolicyEmployee*** *have the same identifiers of a Surrogate Key. There is the need to resolve such a conflict, in the algorithm, by preserving the surrogate key identifier in the preferred data mart and re-assigning a new surrogate key identifier for the non-preferred data mart(s).* ■

Entity Representation Conflicts. These conflicts arise as a result of the multiple representations of the same real-world entity in the merged dimension by the different identifiers. This occurrence is traced to different representations of surrogate key identifiers from different dimensions for the same real-world entity in the merged dimension. A proposed resolution measure, outlined in the merged algorithm, will be to perform a de-duplication of the conflicting entities. This is achieved by preserving the entity from the preferred data mart as the sole representation of the real-world entity in the merged dimension.

Attribute Property Type Conflicts. These forms of conflicts occur as a result of the existence of different attribute property values from the integrating attributes into a merged attribute. In reference to Example 6, in integrating dimensions *DimPolicyEmployee and DimInsuredPolicyEmployee*, a merged attribute for *DimPolicyEmployee.HireStatus* and *DimInsuredPolicyEmployee.EmployeeStatus* attributes will hold a data type value of, say *varchar(1)*, being the *UNION* of integrating attribute data types for *char(1)* and *bit* data types from *DimPolicyEmployee.HireStatus* and *DimInsuredPolicyEmployee.EmployeeStatus*, respectively. We resolve these conflicts by using the attribute data types as defined in the mapping model.

VIII. IMPLEMENTATION AND EVALUATION

In this section, we discuss the implementation and evaluation work based on the integration methodology and formulated merge algorithm. We present our implementation framework and the procedures, and we discuss and analyze the evaluation results.

A. Implementation

We describe our implementation framework of various techniques and processes needed in producing the output of a single consolidated data warehouse. This sub-section focuses on the experimental setup, the datasets used in the experiments, as well as the practical procedures we performed based, on the proposed integration methodology addressed in Sections III, IV, V and VI.

Experimental Setup and Data Sets. We implemented our methodology using 2 different data warehouses, from Insurance and Transportation data sets. The Insurance data consisted of 2 data marts. These were Policy and Claims data marts. Their schema structure and instance data are described. The Policy and the Claims data marts contained 7 and 10 Dimension Table schemas, respectively. These dimensions were referentially connected to a single Fact Table schema. Each fact table schema had a Degenerate Dimension (DD) attribute of a Policy Number and a fact or measure attribute of Policy Transaction Amount. The Policy fact table schema contained instance data of 3,070 tuples of data, whilst the Claims fact contained 1,144 tuples of data. Both data sets had 6 corresponding entity representation in the dimension tables, whilst the Claims data mart had 3 other non-corresponding dimensions.

The Transport data set, on the other hand, contained 3 data marts. These were Frequent Flyer, Hotel Stays, and Car Rental data marts. All the data marts had 3 conformed dimensions; namely, Customer, Date, and Sales Channel. These dimensions were complemented with a number of non-corresponding and unique dimensions in each of the data marts. Their Fact Tables contained 7257, 2449, 2449 tuples of data for Frequent Flyer, Hotel Stays, and Car Rental, respectively. Each of the Fact Tables also contained 7, 6, and 5 facts or measures for Frequent Flyer, Hotel Stays, and Car Rental, respectively. All the source data marts had their permanent repository stored in Microsoft SQL Server DBMS [45]. Each entity representation in the dimensions was identified by unique surrogate key and based on clustered indexing.

Hybrid Schema Matching. The schema matching and mapping models discovery procedural steps were implemented using IBM Infosphere Data Architect [22] [23] [41]. This tool incorporated the schemas of the data mart source repositories, together with their contained instance data. The schema matching step was implemented using the set of algorithmic techniques incorporated in the application software. The schema-based algorithmic techniques that were adopted are Lexical Similarity and Semantic Names, where as the instance-based algorithmic techniques were Signature, Distributions and Regular Expressions. The algorithms were configured by sequentially manipulating the order of execution, configuration of rejection threshold, sampling size and sampling rate. The manipulations of these configurations for finding mapping correspondences were based on an iterative procedure of inspection.

Figure 3 illustrates the derivation of semantically correct matching candidates to establish mapping correspondences between the attributes of *DimPolicyTransactionType.PolicyTransactionTypeKey*, *DimPolicyTransactionType.PolicyTransactionId*, and *DimPolicyTransactionType.TransactionCodeName* of *DimPolicyTransactionType* dimension schema to the *DimClaimTransactionType.ClaimTransactionCode* attribute of *DimClaimTransactionType* dimension schema. In Figure 3, the blue-coloured mapping correspondences represent the chosen semantically correct matching candidate, where *DimPolicyTransactionType.PolicyTransactionId* attribute corresponds to the *DimClainTransactionType.ClaimTransactionCode* attribute. On the other hand, the red-coloured mappings represent the semantically incorrect matching candidates of *DimPolicyTransactionType.PolicyTransactionTypeKey* and *DimPolicyTransactionType.TransactionCodeName*, which are ignored as part of user validation by inspection. Moreover, the yellow-coloured mappings represent the correspondences that were generated for each of the dimensions, as a result of the application of the schema matching algorithms.

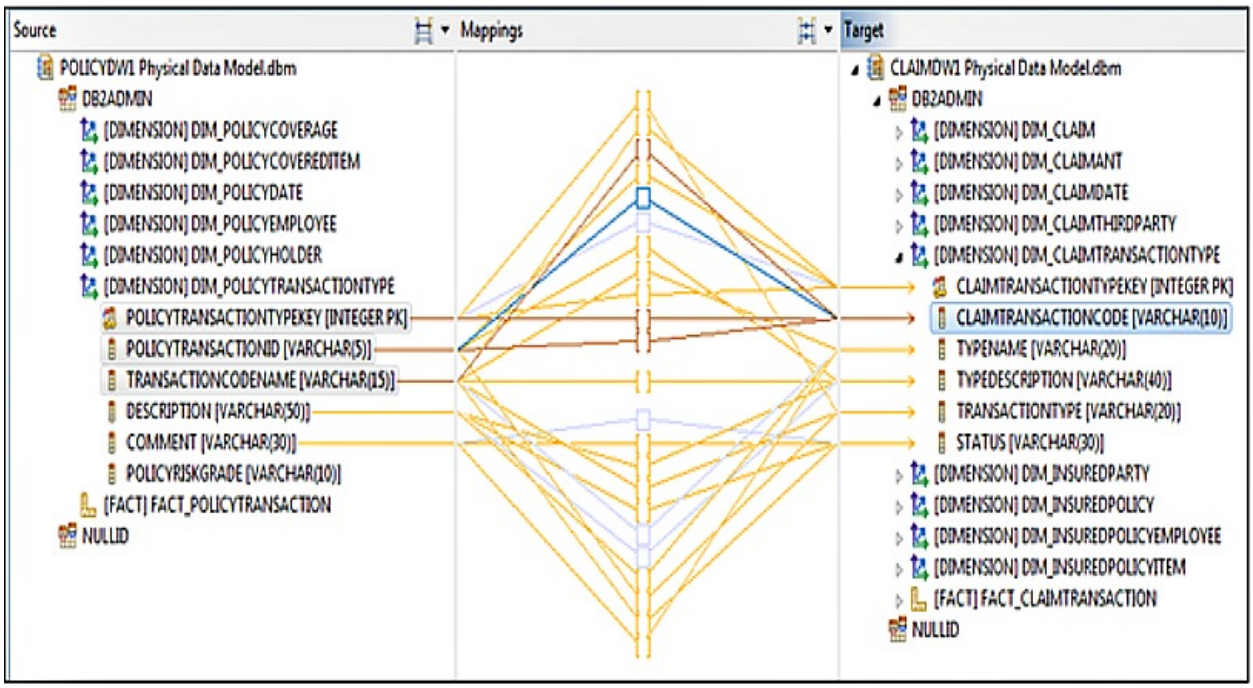


Figure 3. Hybrid Schema Matching

TABLE I. SUMMARY OF PARAMETIZED CONFIGURATIONS FOR SCHEMA MATCHING ALGORITHMS

Matching Algorithm/ Configuration Option	Rejection Threshold	Thesaurus Option	Sampling Size (Rows)	Sampling Rate (%)
1. Lexical Similarity	0.6	Not Applicable	Not Applicable	Not Applicable
2. Semantic Name	0.5	Is Applicable; But not configured	Not Applicable	Not Applicable
3. Signature	0.8	Not Applicable	150	30
4. Distributions	0.8	Not Applicable	100	20
5. Regular Expressions	0.9	Not Applicable	100	30

When generating mapping correspondences for the fact and dimension table attributes, various configuration manipulations of algorithms are performed on the discovery function. The parameters used in configuring the algorithms were Rejection Threshold, Thesaurus Option, Sampling Size, and Sample Rate. The Rejection Threshold parameter was configured with different adjustments for both the schema- and instance-based algorithms. The Thesaurus Option parameter was applicable to the Semantic Name algorithm. The Sampling Size and Sampling Rate parameters were applicable to the instance-based algorithms. We summarize the parameterized configuration of the algorithms adopted in TABLE I.

Mapping Models Discovery. In the mapping models discovery step, the adoption of GLAV mappings enabled the inclusion of all attributes for each mapping formulation of fact and dimension table attributes. Moreover, complex transformation expressions were derived for multi-cardinality mappings.

An illustration of multi-cardinality mapping relationship is displayed in Figure 4. In Figure 4, there is a mapping discovery and modelling between the attributes of *DimPolicyHolder* and *DimInsuredParty* dimensions. These mappings are indicated by the grey lines connecting attributes from *DimPolicyHolder* to *DimInsuredParty* dimensions. More specifically, a selected mapping relationship of the *DimInsuredParty.FullName* attribute is modelled onto 2 other attributes; namely, *DimInsuredParty.FamilyName* and *DimInsuredParty.GivenName*.

We therefore, defined a complex transformation expression, as in Equation (1), in the mapping relationship already established between these dimension attributes.

$$DimInsuredParty.FullName = DimPolicyHolder.FamilyName + DimPolicyHolder.GivenName \quad (1)$$

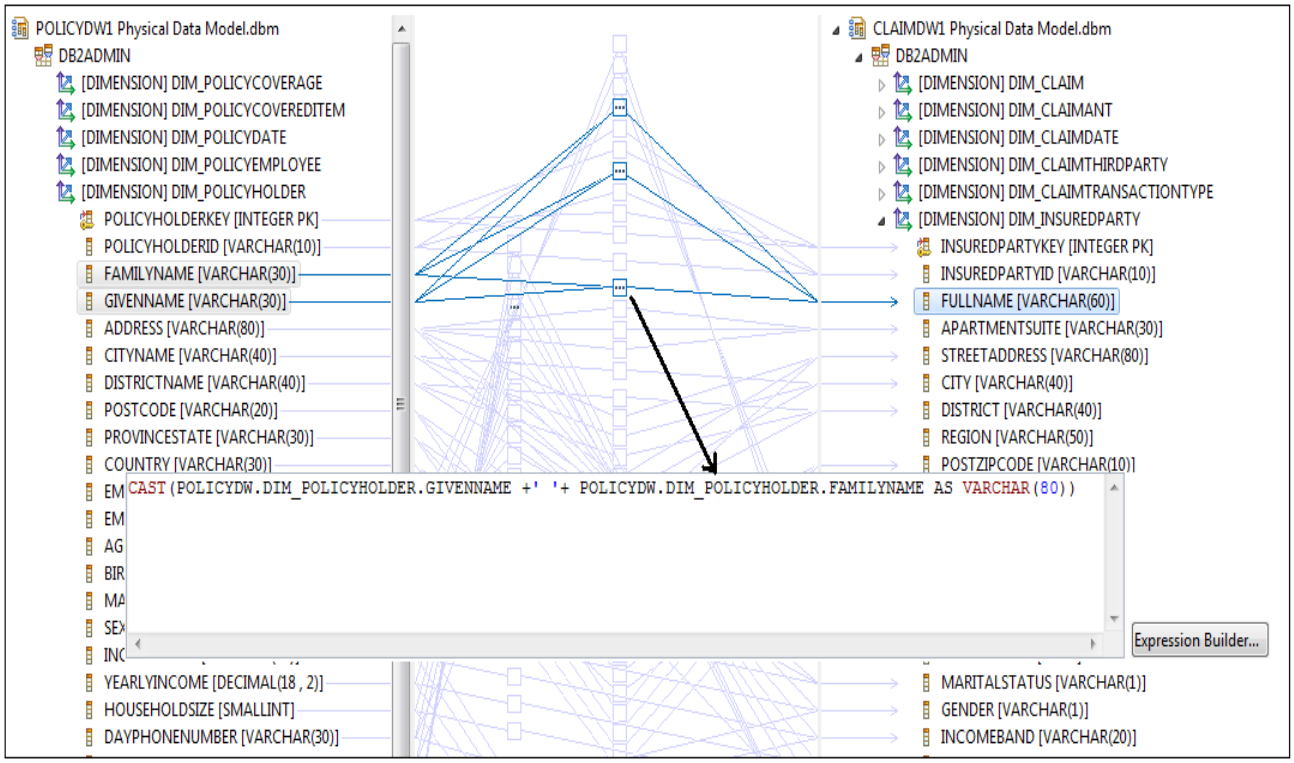


Figure 4. Mapping Models Discovery

Other forms of mapping properties that were defined in the modelling are expressive characterization of relationship cardinality, attribute semantic representation, and attribute data type representation, amongst others. In terms of the relationship cardinality, an equality or similarity mapping cardinality type was defined. To express the attribute semantic representation, a definition of the supposed merged attribute name was specified, where possible. Regarding attribute data type representation, a supposed merge data type was defined and this served as a union data type for the merging attributes. A procedural output in a *Comma Separated Values* (CSV) file format was later generated, which contained the mapping definitions based on the tables, their attributes, and the attribute property values from each of the data marts

Formulated Merge Algorithm. The formulated merge algorithm was implemented with the availability of the mapping models and the source data marts as inputs. The implementation was programmed using Microsoft Visual C# .Net Integrated Development Environment (IDE) with 8029 lines of code from the Entity classes, Business Logic classes, Utility classes, and program control code. Stored procedures were implemented in the Microsoft SQL Server permanent repository, and these served as transaction processing medium between the data repository and the entity and business logic classes in the programming IDE.

Query Processing and Analyses. The analyses of the repository data, of both the integrating source data marts and the generated single consolidated data warehouse, were performed using IBM Cognos Business Intelligence [46] application software. The software enabled the possibility of processing queries on the instance data, in the form of report generation.

B. Evaluation

Our evaluation analyses were primarily based query processing on the single consolidated data warehouse in relation to the integrating data marts. We compared the outputs of the query processing from both the data marts and the generated data warehouse. We first ran a formulated query the data marts, and afterwards ran the same query on the generated data warehouse. Based on these processes, we are able to effectively compare the results from the data marts and the single consolidated data warehouse.

Evaluation Criteria and Analyses. We evaluate the outcome of the experiments performed based on a set of criteria from the guidelines proposed by Pedersen et al. [44]. We performed a gap analysis on their study and adapted correctness of data values, dimensionality hierarchy, and rate of query processing, as criteria.

The metrics that we used in evaluating these criteria for query processing were *recall*, *precision*, and *accuracy* 0. Recall is computed by the number of tuples retrieved from a data mart divided by the number of tuples that should have been retrieved from the generated data warehouse from each original data mart. Precision is computed by the number of tuples retrieved from a data mart divided by the number of

tuples that were retrieved from the single consolidated data warehouse, per the data mart. Accuracy is determined by the degree of validity or exactness of the data values generated from a query posed to the data warehouse in comparison to the data values retrieved from a data mart.

For recall, an evaluation of 100% was trivially attained and verified. The verification was based on the assertion that the formulated merge algorithm fulfilled the MCRs of *Measure and Attribute Entity Preservation* and *Tuple Containment Preservation*.

Precision evaluation was very important, as it measured the proportion of relevant and non-relevant tuples that were retrieved based on a formulated query. This presents an insight into the composition of our merged data warehouse, in terms of the level of integration of related data from multiple sources. Deducing from the precision values, a higher rate was attained for all formulated queries that were posed against the data warehouse. For cases of dimensions that were only related to some specific data marts, a formulated query yielded a very high precision rate. This was as a result of the retrieval of few non-relevant tuples. An example query was, "*What insurance claimant employment type receives the most claims processed for the current Calendar Season*"? Conversely, for queries on dimensions that related or corresponded to all data marts, an average precision rate was observed where a considerable number of non-relevant tuples were retrieved in reference to a particular data mart. An example query was, "*What type of Policy Coverage is most popular? What are the trends since the 2nd Calendar Quarter.*"

Figures 5 and 6 show the precision evaluation for Insurance and Transportation data warehouses, respectively. In Figure 5, an average rate of 86% was achieved for the queries posed to dimensions related to the Claims data mart. The precision rate increases significantly with an increase in the tuples in these dimensions, as more relevant tuples are generated. This is evident in queries 1 to 7. In terms of corresponding dimensions for all data marts, processed queries generated an average rate of 51% and 49% for Claims and Policy data marts, respectively, as highlighted in queries 8 to 12.

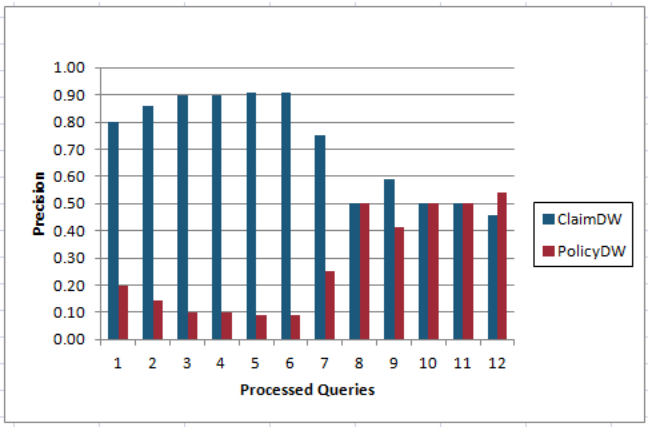


Figure 5. Precision for Insurance Data Set

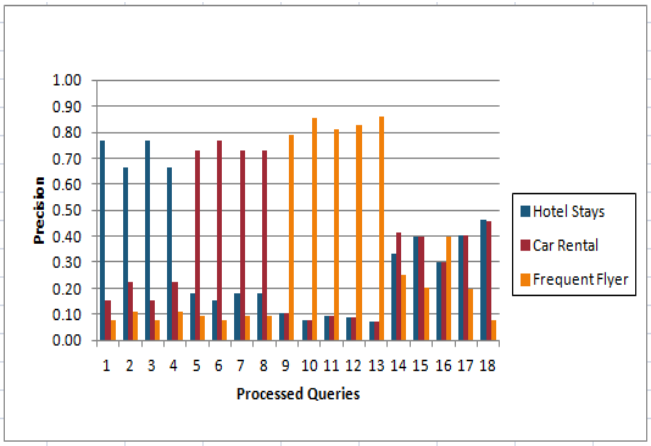


Figure 6. Precision for Transportation Data Set

In Figure 6, an average precision rate of 72%, 74%, and 83% were attained for Hotel Stays, Car Rental, and Frequent Flyer data marts, respectively, for the set of formulated queries posed. Queries 1 and 3 for Hotel Stays, 6 for Car Rental, and 10, 12, 13 for Frequent Flyer data marts performed creditably well as a result of the higher containment of tuples to the attributes being retrieved for the formulated queries posed. Moreover, in terms of queries posed to corresponding dimensions, an average precision rate of 38%, 40%, and 23% was attained for Hotel Stays, Car Rental, and Frequent Flyer data marts, respectively. This is depicted in queries 14 to 18. It would be realized that this average rate for the Transportation data set is quite lower than that attained in respect to the Insurance data set. This is based on the claim that an increase in the number of data marts for integration is inversely proportional to the precision rate of queries for the respective data marts. This assertion is due to the distributive proportionality of tuples per each dimension of the corresponding data marts. Additionally, the attributes involved in the formulated query for these dimensions also enforces on this assertion.

In summary, the average precision rates analyzed are able to provide the user with details regarding the proportion of the data in the merged data warehouse that originate from a specific data source. This holds important practical value, for data warehouse practitioners, who want to be able to have statistics regarding the composition of the merged data.

In terms of accuracy, we achieved a 100% return rate of valid and exact data values from the data warehouse, in comparison to each individual data mart. This was affirmed based on the merge algorithm fulfilling MCRs of *Tuple Containment Preservation* and *Measure and Attribute Entity Preservation*. Additionally, the adoption of GLAV mapping model enabled the processing of exact and sound queries on the data warehouse.

Query Processing Rate. We also analyzed the rate of query processing to ensure that queries posed to the data warehouse are of optimal rate. With an integration of instance data from the data marts, a considerable volume of expected data cannot be overemphasized in the data

warehouse. We recorded the query response time for an average of 20 query executions for each of the data sets. These queries were processed on a single 3.20 GHz processor with a 4 GB of RAM.

Our evaluation of the processed queries showed that the queries generally ran at almost the same rate or slightly higher than when posed against the data mart sources. The query execution durations for the data marts and data warehouses for the Insurance and Transportation data sets are shown in Figure 7 and 8, respectively.

In Figures 7 and 8, it can be generally deduced from display that the data values that the query rate for the data warehouses were appreciable taking note of the compared values generated from the data marts. In certain cases, such as queries 7 and 8, in Figure 7, the rates were a bit higher due to higher level of aggregation and increased number dimension attributes involved in data values retrieved. Queries 6 and 11 recorded lower query rates because of the low quantity of attributes, as well as tuple data values, in the formulation of the answer to the query. Additionally, in Figure 8, similar observation was realized on queries 6, 14, and 16 where the query processing rate is a bit higher in comparison to the others. We also observed a lower rate of query rates for queries 4, 10, 13, and 19, which inferred a very good composition of merged tables and attributes and their contained data instance tuple values.

We further computed the variance of the average query rate per data mart as it differs quantitatively from the consolidated data warehouse. A deduction observation was ascertained, where a lower quantity of tuples of instance data values to be retrieved during query processing lead to an increase in the variance, and vice versa. This is due to the fact that an increase in the number of data marts, and resultant increase in data instance tuples, increases the rate of data retrieval, per data mart analysis in relation to the single consolidated data warehouse.

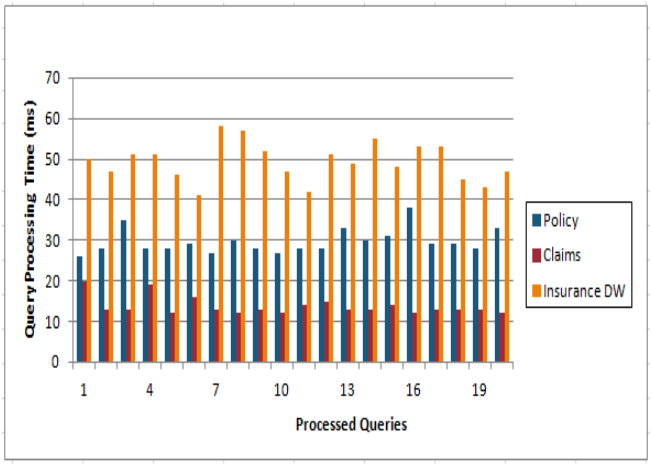


Figure 7. Query Processing Rate for Insurance Data Set

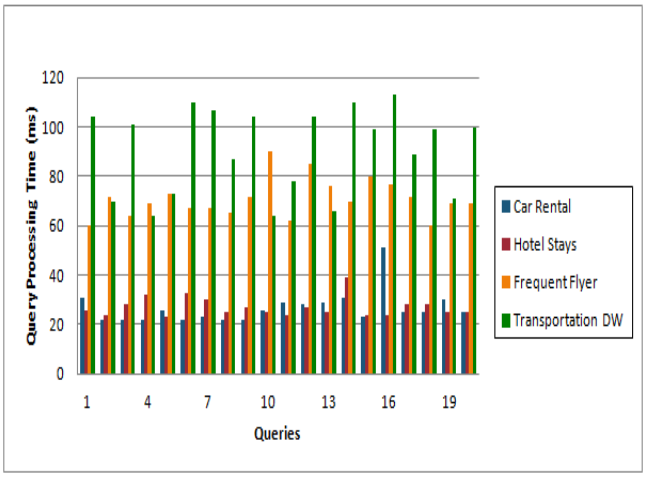


Figure 8. Query Processing Rate for Transportation Data Set

TABLE II. SUMMARY OF AVERAGE QUERY RESPONSE TIME AND VARIANCES

Data Set	Average Query Response Time and Variances		
	Data Mart / Data Warehouse	*Avg. Query Response (ms)*	*Variance From Integrated Data Warehouse (ms)*
Transportation	Car Rental	26.70	63.95
Transportation	Hotel Stays	27.10	63.55
Transportation	Frequent Flyer	70.95	19.70
Transportation	DataWarehouse	90.65	0.00
Insurance	Policy	29.65	19.60
Insurance	Claim	13.75	35.50
Insurance	DataWarehouse	49.25	0.00

An observation of the Claims data mart, in Figure 7 and TABLE II. reveals that the variance of 35.50 was higher because of the lower query rate of the integrating data mart. Moreover, in Figure 8 and TABLE II. the Hotel Stays and Car Rental data marts rather had a higher variance of 63.55 and 63.95, respectively, as their query rates were lower because of the lower quantity of data instance tuples.

We present a summary of the variances in the average query response time (in milliseconds) for the data marts in comparison to their respective data warehouses in TABLE II.

IX. COMPARISON TO OTHER APPROACHES

There have been minimal studies in this area of multidimensional data integration, in particular to the generation of a single consolidated data warehouse. These approaches present significant contributions with regards to element mappings and algorithms. In comparison, our approach addresses the integration problem from an important concept of model management. We discuss a number of these approaches and comparatively explain how our methodology performs better.

A. Dimension Compatibility and Heterogeneous Multidimensional Integration

Cabibbo and Torlone in their series of studies [31] [32] [33], address the problem of data integration in relation to multidimensional databases (data marts). In their work [31] [33], they introduce fundamental assertions of dimension algebra and dimension compatibility. Their work highlights different forms of heterogeneities that are existent in dimension tables. Their attempt to address these heterogeneities lead them to introduce a novel theoretical concept of dimension algebra. This concept enables the selection of relevant portions of a dimension for integration. The dimension algebra is basically based on 3 main operators; namely, selection, projection, aggregation.

The authors in [31] [33] also introduce the concept of dimension compatibility. Dimension compatibility outlines the retrieval of common dimension information based on the characterization of general properties. These general properties were outlined as; level equivalence, dimension equivalence, dimension comparability, and dimension intersection. The compatibility property of dimensions was then used as a platform to perform drill-across queries over the autonomous data marts, and aid in hierarchical aggregation of instance data. As part of their study, the authors [31] [32] [33] use the fundamental intuitions to propose 2 different approaches to the problem of integration of multidimensional databases; namely, loosely coupled integration and tightly coupled integration. They introduced concepts and algorithms, and stipulated a number of desirable properties for dimension matching; namely, coherence, soundness, and consistency.

B. Inferred Aggregation in Hierarchies

Riazati et al. [34] propose a solution for integration of data marts where they infer aggregations in the hierarchies of the dimension tables existent in the multidimensional databases. In their work, they attempted formulating a computation on minimal directed graph from the instance data. These inferred hierarchies are then used to perform roll-up relationships between levels and to ensure the summarizability of data. They further use the assertion of dimension compatibility introduced in [31] [32] [33] to develop algorithms, which in turn are used for the integration of data marts.

C. Methodology Comparisons & Evaluation

The existing approaches to multidimensional instance schema data integration addressed in [31] [32] [33] [34] explain important concepts that need to be discussed when incorporating several data marts into a single consolidated data warehouse. On the contrary, these techniques and methodologies are inadequately enough in the handling of more complex characteristics of the fact or dimension tables and their attributes. We address the shortcomings of these approaches, and highlight the enhanced ways of handling such issues through our methodology approach using the concept of model management.

Firstly, the approaches by the authors in [31] [32] [33] fail to address the issue of mapping models, although propositions of the general properties regarding the characterization of dimension compatibility seems to handle this concept. Our approach, however, adopts a first-order mapping modelling formalism, which better expresses the attribute correspondences. As a result, issues of data exchange and transformation for dissimilar and multicardinality attributes are expressed efficiently.

Secondly, the previous approaches do not lay out a precise schema merge algorithm. Descriptions of algorithms for deriving the common information between dimensions and for merging were put forward in [32] and other literatures so far. But these algorithms are inconclusive enough to solve the complex representations of schema and data instances. Our approach offers a complete formulated algorithm for integrating multidimensional data models based on star schema models.

Thirdly, conflict management relating to identification and resolution are not completely addressed by the authors in their approach. In the literature [33], the properties that underlie and establish the dimension compatibility criteria seem to partially solve the likely to occur conflicts that could be encountered in the dimensions. But these properties in their entirety fail to totally resolve such prominent conflicts during integration. Our methodology outlines a definite set of likely to occur conflicts and their resolution measures in relation to the instance schema and instance data values.

Fourthly, technical qualitative requirements, which serve to highlight the properties that a generic integrated schema should possess were addressed by the authors in [2][28]. A careful study of the specific approaches for multidimensional data integration attempted by the authors in [31] [32] [33] [34] seem not to have specified requirements for integration. A number of requirements were generally attempted by the authors in [32]. They proposed of coherence, soundness and consistency as measures for compatible dimension matching; but these are inconclusive in the larger scale of integrating schema and data instances. Our methodology approach proposes a complete set of requirements for multidimensional integration to handle the varied properties and constraints of multidimensional data models.

We present a comparative analysis and evaluation of the proposed methodology in line with other approaches in TABLE III. This tabular analysis summarizes the discussions regarding methodology approaches presented in the literature, and outlines the merits of our proposed methodology over the other approaches.

X. CONCLUSION

This paper presents a methodology for the merging of multidimensional data models using star schemas instances. We addressed extensively the methodologies and algorithms adopted in finding mapping correspondences between the elements attributes of the fact and dimension tables for the data marts.

TABLE III. QUALITATIVE ANALYSIS OF PROPOSED METHODOLOGY AND OTHER APPROACHES

Methodology Approach / Analysis Criteria	**(1) Proposed Integration Methodology**	**(2) *Cabibbo and Torlone*** [31] [32] [33] **- Dimension Compatibility and Heterogeneous Multidimensional Integration**	**(3) *Riazati et al.*** [34] **– Inferred Aggregation in Dimension Hierarchies**
Mapping Models Discovery and Modelling	Adopts a first-order GLAV mapping model, which offers effective data translation and data exchange functions	Introduces dimension compatibility for attribute mappings, but does not present complete mapping modelling and the handling of attribute relationships types	Methodology extends on the previous notions of dimension compatibility in (2); does not lay out precise mapping modelling
Formulated Merge Algorithm	Presents a complete merge algorithm that handles varied characteristics of both schema and data instances from heterogeneous data sources	Presents sets of algorithms that involves drill-across queries between dimensions instance schema attributes; but methods are inconclusive for varied properties of instances of schema attributes and data	Proposes algorithms for inferring partial order of attributes, and for identifying hierarchy levels and roll-ups. These algorithms are based on only schema instances
Conflict Identification and Resolution	Identifies likely to occur conflicts and proposes complete resolution measures in the element attributes and their properties	Conflict management is not clearly addressed by the authors. Attempts of using dimension algebra and dimension compatibility is not sufficient to handle frequently observed conflicts	Methodology does not precisely outline conflict identification and resolution measures for the schema instances of tables and their attributes
Technical Qualitative Requirements	Proposition of requirements to handle the integration of varied characteristics of schema and data instances; to generate an merged data warehouse, and for effective query processing	Proposition of *Coherence*, *Soundness*, and *Consistency*; as measures for compatible dimension matching, but the requirements are inconclusive to handle varied properties of schema and data instances	Methodology does not propose qualitative requirements; but adopts and extends the properties outlined in methodology (2) to infer attribute matchings and aggregations in the hierarchies

Here, we adopted a hybrid schema matching methodology for finding mapping correspondences. We also outlined the adoption of first-order GLAV mapping models and their attribute relationship characterization of equality and similarity mappings. Moreover, we addressed the handling of mapping modelling constraints in the form of functional dependencies in the dimensions. We formulated a merge algorithm for integrating disparate data marts into a single consolidated star schema data warehouse.

Furthermore, we addressed the semantics of query processing on the single consolidated data warehouse taking cognizance of the aggregations in hierarchy and summarizability of data instance values for the hierarchies. We identified and outlined the resolution of frequently observed conflicts that are encountered when merging data marts. To this end, we outlined the satisfaction of technical merge correctness requirements for integrating data marts into a data warehouse.

Finally, we compared our methodology of integrating schema and data instances as against other approaches. We outlined the merits and suitability of our approach for delivering an enterprise-wide single consolidated data warehouse from a number of disparate data marts.

The analyses of our evaluation showed that the rates of recall, precision and accuracy of the data values retrieved from the generated data warehouse are high and noticeable. We specifically analyzed the precision of queries in different situations of query processing for corresponding or non-corresponding dimensions from the integrating data marts. We also analyzed the rate of query processing on the single consolidated data warehouse as compared to the individual data marts. We observed that with an increase in the number of data marts, and more specifically, an increase in the data instance tuples the variance of query processing for the concerned data marts decreases considerably. Our approach, thus, provides data warehouse researchers and practitioners with procedures, criteria, and exact measures as to how successful an integration process is achieved.

A number of future research directions remain. The incorporation of data mart level integrity constraints into the data warehouse needs to be investigated further. We also envisage the extension of the methodology to handle snowflake and fact-constellation multidimensional data models.

REFERENCES

[1] M. Mireku Kwakye, I. Kiringa, and H. L. Viktor, "Merging Multidimensional Data Models: A Practical Approach for Schema and Data Instances," In Proceedings of the 5th International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA), 2013, pp. 100-107.

[2] R. A. Pottinger and P. A. Bernstein, "Merging Models Based on Given Correspondences," In Proceedings of the 29th International Conference on Very Large Data Bases (VLDB), 2003, pp. 826-873.

[3] M. Lenzerini, "Data Integration: A Theoretical Perspective," In Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 2002, pp. 233-246.

[4] P. A. Bernstein and S. Melnik., "Model Management 2.0: Manipulating Richer Mappings," In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2007, pp. 1-12.

[5] S. Melnik, "Generic Model Management: Concepts and Algorithms," Springer Lecture Notes in Computer Science (LNCS), 2004, pp. 2967.

[6] P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger, "A Vision of Management of Complex Models," In Proceedings Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2000, vol. 29, no. 4, pp. 55-63.

[7] S. Melnik, "Model Management: First Steps and Beyond," In Proceedings of the 11th Symposium of the GI Department, Database Systems in Business, Technology and Web, (BTW), 2005, pp. 455-464.

[8] M. N. Gubanov, P. A. Bernstein, and A. Moshchuk, "Model Management Engine for Data Integration with Reverse-Engineering Support," In Proceedings of the 24th International Conference on Data Engineering (ICDE), 2008, pp. 1319-1321.

[9] E. Rahm and P. A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," Very Large Data Bases (VLDB) Journal, 2001, vol. 10, no. 4, pp. 334-350.

[10] P. Shvaiko and J. Euzenat, "A Survey of Schema-based Matching Approaches," Journal of Data Semantics IV, vol. 3730, pp. 146-171, 2005, doi:10.1007/11603412_5.

[11] P. Shvaiko, "A Classification of Schema-based Matching Approaches," In Proceedings of the Meaning Coordination and Negotiation Workshop at the 3rd International Semantic Web Conference (ISWC), 2004.

[12] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching," In Proceedings of the 18th International Conference on Data Engineering (ICDE), 2002, pp. 117-128.

[13] H. H. Do and E. Rahm, "COMA: A System for Flexible Combination of Schema Matching Approaches," In Proceedings of 28th International Conference on Very Large Data Bases (VLDB), 2002, pp. 610-621.

[14] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic Schema Matching with Cupid," In Proceedings of 27th International Conference on Very Large Data Bases (VLDB), 2001, pp. 49-58.

[15] W-S. Li and C. Clifton, "SEMINT: A Tool For Identifying Attribute Correspondences In Heterogeneous Databases Using Neural Networks," Elsevier Science. Data and Knowledge Engineering (DKE), 2000, vol. 33, no. 1, pp. 49-84.

[16] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy, and P. Domingos, "iMAP: Discovering Complex Mappings between Database Schemas," In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2004, pp. 383-394.

[17] M. A. Hernandez, R. J. Miller, and L. M. Haas, "Clio: A Semi-Automatic Tool For Schema Mapping," In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2001, pp. 607.

[18] R. J. Miller, M. A. Hernandez, L. M. Haas, L-L. Yan, C. T. H. Ho, R. Fagin, and L. Popa, "The Clio Project: Managing Heterogeneity," ACM SIGMOD Record, 2001, vol. 30, no. 1, pp. 78-83.

[19] A. Y. Halevy and J. Madhavan, "Corpus-Based Knowledge Representation," In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI), 2003, pp. 1567-1572.

[20] J. Berlin and A. Motro, "Database Schema Matching Using Machine Learning with Feature Selection," In Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE), 2002, pp. 452-466.

[21] A. Islam, D. Z. Inkpen, and I. Kiringa, "Applications of Corpus-based Semantic Similarity and Word Segmentation to Database Schema Matching," Very Large Data Bases (VLDB) Journal, 2008, vol. 17, no. 5, pp. 1293-1320.

[22] M. A. Hernandez, L. Popa, C. T. H. Ho, and F. Naumann, "Clio: A Schema Mapping Tool for Information Integration," In Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN), 2005, pp. 11.

[23] R. Fagin, L. M. Haas, M. A. Hernandez, R. J. Miller, L. Popa, and Y. Velegrakis, "Clio: Schema Mapping Creation and Data Exchange," Conceptual Modelling: Foundations and Applications, 2009, pp. 198-236.

[24] D. Kensche, C. Quix, X. Li, Y. Li, and M. Jarke, "Generic Schema Mappings for Composition and Query Answering," Elsevier Science. Data and Knowledge Engineering (DKE), 2009, vol. 68, no. 7, pp. 599-621.

[25] R. A. Pottinger, "Database Schema Integration," Encyclopedia of GIS, 2008, pp. 226-231.

[26] P. A. Bernstein and E. Rahm, "Data Warehouse Scenarios for Model Management," In Proceedings of the 19th International Conference on Conceptual Modeling (ER), 2000, pp. 1-15.

[27] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati, "Data Integration in Data Warehousing," International Journal of Cooperative Information Systems (IJCIS), 2001, vol. 10, no. 3, pp. 237-271.

[28] R. A. Pottinger and P. A. Bernstein, "Schema Merging and Mapping Creation for Relational Sources," In Proceedings of the 11th International Conference on Extending Database Technology (EDBT), 2008, pp. 73-84.

[29] N. Rizopoulos and P. McBrien, "Schema Merging Based on Semantic Mappings," In Proceedings of the 26th British National Conference on Databases (BNCOD), 2009, pp. 193-198.

[30] C. Quix, D. Kensche, and X. Li, "Generic Schema Merging," In Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE), 2007, pp. 127-141.

[31] L. Cabibbo and R. Torlone, "On the Integration of Autonomous Data Marts," In Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM), 2004, pp. 223-231.

[32] L. Cabibbo and R. Torlone, "Integrating Heterogeneous Multidimensional Databases," In Proceedings of the 17th International Conference on Scientific and Statistical Database Management (SSDBM), 2005, pp. 205-214.

[33] L. Cabibbo and R. Torlone, "Dimension Compatibility for Data Mart Integration," In Proceedings of the 12th Italian Symposium on Advanced Database Systems (SEBD), 2004, pp. 6-17.

[34] D. Riazati, J. A. Thom, and X. Zhang, "Inferring Aggregation Hierarchies for Integration of Data Marts," In Proceedings of the 21st International Conference on Database and Expert Systems Applications (DEXA), 2010, pp. 96-110.

[35] IBM, *IBM Infosphere Data Architect 7.5.3.0: Finding Relationships.* [Online]. Available: http://publib.boulder.ibm.com/infocenter/idm/v2r1/index.jsp?topic=/com.ibm.datatools.metadata.mapping.ui.doc/topics/iiymdadconfiguring.html. Retrieved: 2014.05.31.

[36] E. Deza and M. M. Deza, "Euclidean Distance," Encyclopedia of Distances, Springer, 2009, pp. 94.

[37] S. Craw, "Manhattan Distance," Encyclopedia of Machine Learning, Springer, 2010, pp. 639.

[38] M. Dash and H. Liu, "Feature Selection for Classification," Intelligent Data Analysis, 1997, vol. 1, no. 3, pp. 131–156.

[39] B. Ten Cate and P. G. Kolaitis, "Structural Characterizations of Schema-Mapping Languages," In Proceedings of the 12th International Conference on Extending Database Technology (ICDT), 2009, pp. 63-72.

[40] M. Friedman, A. Levy, and T. Millstein, "Navigational Plans for Data Integration", In Proceedings of the 16th National Conference on Artificial Intelligence and 11th Conference on Innovative Applications of Artificial Intelligence (16. AAAI/ 11. IAAI), 1999, pp. 67-73.

[41] IBM, *IBM Infosphere Data Architect 7.5.3.0.* [Online]. Available: http://www-01.ibm.com/software/data/optim/data-architect. Retrieved: 2014.05.31.

[42] Microsoft, *Microsoft BizTalk Mapper*. [Online]. Available: http://msdn.microsoft.com/en-us/library/ee253382(v=bts.10).aspx. Retrieved: 2014.05.31.

[43] R. Kimball, M. Ross, W. Thornthwaite, J. Mundy, and B. Becker, "The Data Warehouse Lifecycle Toolkit," John Wiley and Sons, 2nd Edition, 2008, ISBN-10: 0470149779.

[44] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson, "A Foundation for Capturing and Querying Complex Multidimensional Data," Elsevier Science. Information Systems (IS), 2001, vol. 26, no. 5, pp. 383-423.

[45] Microsoft, *Microsoft SQL Server Database Management System*. [Online]. Available: http://www.microsoft.com/en-us/sqlserver/default.aspx. Retrieved: 2014.05.31.

[46] IBM, *IBM Cognos Business Intelligence 10.2.0.* [Online]. Available: http://www-03.ibm.com/software/products/en/business-intelligence. Retrieved: 2014.05.31.

M. Junker, A. Dengel, and R. Hoch, "On the Evaluation of Document Analysis Components by Recall, Precision, and Accuracy," In Proceedings of the 5th International Conference on Document Analysis and Recognition (ICDAR), 1999, pp. 713-716.

XI. APPENDIX

MERGE ALGORITHM PROOF OF CORRECTNESS

A. Preliminaries

In this section, we provide an outlined proof of correctness of the formulated merge algorithm, which establishes query processing on the single consolidated data warehouse.

Definition 9. (Certain Query): A *Query*, $\mathcal{Q}$ is said to be *Certain* for all Instances, $\mathcal{I}$ and Properties, $\mathcal{P}$ of a Multidimensional Database, $\mathcal{MD}$ iff $\mathcal{Q} \vDash \mathcal{I}$, such that $\mathcal{I} \subseteq \mathcal{MD}$ and $\mathcal{Q}$ satisfies $\mathcal{P} \in \mathcal{MD}$ ■

Definition 10. (Certain Answer): A *Tuple*, $\mathcal{T}$ forming an *Answer* to a certain query, $\mathcal{Q}$ is said to be *Certain* iff $\mathcal{T} \vDash \mathcal{Q}$ for all Instances, $\mathcal{I}$ of Multidimensional Database, $\mathcal{MD}$ and $\mathcal{T}$ fulfils $\mathcal{I} \in \mathcal{MD}$ ■

Let $\mathcal{V} = \{\mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_p\}$ represent an expected set of p tuple variables of certain answers ranging over a set of queries, $\mathcal{Q}$. Let $\mathcal{Q} = \{\mathcal{Q}_1, \dots, \mathcal{Q}_z\}$ represent a set of z possible and *certain queries* likely to be posed to the single consolidated data warehouse. For the *tuple* $\mathcal{V}$ proving a *query* $\mathcal{Q}$ will mean the *tuple* $\mathcal{V}$ computes *certain answers* to the *query* $\mathcal{Q}$ posed on the single consolidated data warehouse.

Theorem 1. (Merge Algorithm): Let $\mathcal{S}$ and $\mathcal{I}$, respectively, represent the schema and data instances of a *Multidimensional Star Schema Model*, $\mathcal{MD}$. Suppose $\mathcal{MD}$ is instantiated in a *Fact*, $\mathcal{F}$ and m number of *Dimensions* $\mathcal{D}_i, \{1 \leq i \leq m\}$ such that $\mathcal{F} = \{\mathcal{S}_\mathcal{F}, \mathcal{I}_\mathcal{F}\}$, $\mathcal{D}_i = \{\mathcal{S}_{\mathcal{D}_i}, \mathcal{I}_{\mathcal{D}_i}\}$. Then, a merge algorithm which accepts n Star Schema Instances, $\mathcal{MD}_\mathrm{j}$, for $\{2 \leq \mathrm{j} \leq n\}$, and *Mapping Correspondences*, $\mathcal{MAP}_{\mathcal{FD}_i}$ as inputs, generates a *Single Consolidated Data Warehouse*, $\mathcal{DW}$ in a worst-case polynomial time complexity, such that $\{\mathcal{S}, \mathcal{I}\} \in \mathcal{DW} \vDash \{\mathcal{Q}, \mathcal{T}\} \in \mathcal{MD}_\mathrm{j}$ ■

B. Proof of Soundness

PROOF. (SKETCH) **Soundness.** We want to show that, if a *tuple* $\mathcal{V}$ can be proven or computed as a certain answer to a posed *certain query* $\mathcal{Q}$ on the single consolidated data warehouse, $\mathcal{DW}$ then *tuple* $\mathcal{V}$ will answer the *certain query* $\mathcal{Q}$.

(⇒)

By use of inductive definition, we assume for an arbitrary *tuple* $\mathcal{V}$ and *certain query* $\mathcal{Q}$, such that the *tuple* $\mathcal{V}$ is computed in n number of steps for query $\mathcal{Q}$. Consequent to this assumption, the *tuple* $\mathcal{V}$ will represent *certain answers* to the *query* $\mathcal{Q}$. This will hold for all data instances of the single consolidated data warehouse generated from this algorithm.

For Steps (2) to (7), it can be inferred that the mapping correspondences between the integrating instance schema table attributes are iterated in finite steps. The single consolidated data warehouse will then be a representation of all instance schema table attributes.

Since instance data values are associated to each attribute of the schema instances. Hence, *certain answers* for *tuple, say* $\mathcal{V}$, is generated for any *query*, say $\mathcal{Q}$, posed to it.

For Step (5), the intuition that only 2 forms of mapping is adopted implies all forms of mapping ambiguities for possible intractability or a worst-case of an undecidability are not expected. In that regard, exact *certain answers* are expected from a posed query for *equality* mapping types. For *similarity* mapping, similar

certain answers for *tuples* are generated. Non-corresponding attributes also help in generating tuples for local instance schema attributes per data mart. As a result, by inductive proposition the correctness in tuple data values is trivially preserved.

For Step (8), the *tuples* that are generated from schema attributes will have properties of being the *UNION* of all integrating attributes. The unified property thus asserts on all the semantics from each of the integrating attributes. Hence, if a *tuple*, say $\mathcal{V}$, is generated for a *query*, say $\mathcal{Q}$, a truth validity can be ascertained such that the *tuple* will represent a *certain answer*. This makes the inference and inductive claims from the earlier premise satisfy and preserve the *soundness* criteria for correctness. ■

C. Proof of Completeness

The proof of completeness is trivially the converse to the proof of soundness and affirms the validation of the intuition proposed for soundness.

PROOF. (SKETCH) **Completeness.** We want to show that, if a tuple $\mathcal{V}$ is a *certain answer* to a *certain query* $\mathcal{Q}$ posed on the single consolidated data warehouse, $\mathcal{DW}$ then the *tuple* $\mathcal{V}$ can be proven to exist. In other words, for any *query* $\mathcal{Q}$ posed we are sure not to miss any *certain answer* from the tuples that can be generated.

($\Leftarrow$)

We begin the proof by the use of *contraposition* hypothesis to show that: If a *tuple*, say $\mathcal{V}$, cannot be computed or does not exist for a *query*, say $\mathcal{Q}$, then the *tuple* $\mathcal{V}$ cannot represent a *certain answer* to the *query* $\mathcal{Q}$.

Let us assume the *tuple* $\mathcal{V}$ cannot be computed or generated for the *query* $\mathcal{Q}$ in the strong sense.

If the *tuple* $\mathcal{V}$ cannot be computed, then we can construct an infinite general set, $\mathcal{V}^*$ of aggregated *tuples*, which will still not form computed *tuples* to answer the *query* $\mathcal{Q}$.

Based on this construction, we can inductively generate a categorization of all forms aggregation of tuples. We enumerate them as $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_q\}$. We then will inductively define a series of different sets of tuples $\mathcal{V}_n = \{\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_n\}$.

We then let the first of the series of *tuple* sets, $\mathcal{V}_0$ represent the arbitrary *tuple* $\mathcal{V}$. As part of the inductive construction, if the union of one series set of a *tuple*, say $\mathcal{V}_k$, and a subsequent aggregation categorization, say $\mathcal{E}_{k+1}$ is a computed tuple to answer *query* $\mathcal{Q}$, then $\mathcal{V}_{k+1} = \mathcal{V}_k$, meaning we have both tuple sets having the same answering semantics.

On the contrary, if the union of a tuple set, say $\mathcal{V}_k$ and a subsequent aggregation categorization, say $\mathcal{E}_{k+1}$ **does not** form a computed tuple needed to answer *query* $\mathcal{Q}$, then $\mathcal{V}_{k+1} = \mathcal{V}_k \cup \{\mathcal{E}_{k+1}\}$, where the new tuple, $\mathcal{V}_{k+1}$ is definitely giving us a different answer from the initial one, $\mathcal{V}_k$.

We then have the general set $\mathcal{V}^*$ representing a union of all the aggregated tuples, $\mathcal{V}_n$ likely to give an answer to the query. It can be deduced that the general set $\mathcal{V}^*$ holds our supposed *tuple* $\mathcal{V}$.

The general set $\mathcal{V}^*$ does not provide enough computed tuples to form a *certain answer* to the posed *certain query* $\mathcal{Q}$. Because if it does answers the query then additional attribute tuples, as well as other complex formula to the aggregations should make it a valid *certain answer* the query.

The general set $\mathcal{V}^*$ is a closure set with attribute tuples and hierarchy aggregations in relation to our supposed *tuple* $\mathcal{V}$ to forming *certain answers* to the query $\mathcal{Q}$. Hence this closure set $\mathcal{V}^*$ exhibits a satisfiability property for a canonical evaluation of being always true, and never false.

With such a satisfiability property, we can say that there is always a truth-like claim on $\mathcal{V}^*$, where all its generated *tuples* are true and anything outside it false. This will make our computed *tuple* $\mathcal{V}$, always true and make the posted *query* $\mathcal{Q}$, false.

This assertion of the *tuple* $\mathcal{V}$ being true and the posed *query* $\mathcal{Q}$ being false does not offer a claim for the computed *tuple* $\mathcal{V}$ validating as a *certain answer* to the posted *query* $\mathcal{Q}$.

Hence, our preceding proposition of *contraposition* is satisfied and valid. ■

www.iariajournals.org

International Journal On Advances in Intelligent Systems

ICAS, ACHI, ICCGI, UBICOMM, ADVCOMP, CENTRIC, GEOProcessing, SEMAPRO, BIOSYSCOM, BIOINFO, BIOTECHNO, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS, ENERGY, COLLA, IMMM, INTELLI, SMART, DATA ANALYTICS

issn: 1942-2679

International Journal On Advances in Internet Technology

ICDS, ICIW, CTRQ, UBICOMM, ICSNC, AFIN, INTERNET, AP2PS, EMERGING, MOBILITY, WEB

issn: 1942-2652

International Journal On Advances in Life Sciences

eTELEMED, eKNOW, eL&mL, BIODIV, BIOENVIRONMENT, BIOGREEN, BIOSYSCOM, BIOINFO, BIOTECHNO, SOTICS, GLOBAL HEALTH

issn: 1942-2660

International Journal On Advances in Networks and Services

ICN, ICNS, ICIW, ICWMC, SENSORCOMM, MESH, CENTRIC, MMEDIA, SERVICE COMPUTATION, VEHICULAR, INNOV

issn: 1942-2644

International Journal On Advances in Security

ICQNM, SECURWARE, MESH, DEPEND, INTERNET, CYBERLAWS

issn: 1942-2636

International Journal On Advances in Software

ICSEA, ICCGI, ADVCOMP, GEOProcessing, DBKDA, INTENSIVE, VALID, SIMUL, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS, IMMM, MOBILITY, VEHICULAR, DATA ANALYTICS

issn: 1942-2628

International Journal On Advances in Systems and Measurements

ICQNM, ICONS, ICIMP, SENSORCOMM, CENICS, VALID, SIMUL, INFOCOMP

issn: 1942-261x

International Journal On Advances in Telecommunications

AICT, ICDT, ICWMC, ICSNC, CTRQ, SPACOMM, MMEDIA, COCORA, PESARO, INNOV

issn: 1942-2601