

International Journal on

Advances in Software



2012 vol. 5 nr. 3&4

The *International Journal on Advances in Software* is published by IARIA.

ISSN: 1942-2628

journals site: <http://www.ariajournals.org>

contact: petre@aria.org

Responsibility for the contents rests upon the authors and not upon IARIA, nor on IARIA volunteers, staff, or contractors.

IARIA is the owner of the publication and of editorial aspects. IARIA reserves the right to update the content for quality improvements.

Abstracting is permitted with credit to the source. Libraries are permitted to photocopy or print, providing the reference is mentioned and that the resulting material is made available at no cost.

Reference should mention:

International Journal on Advances in Software, issn 1942-2628
vol. 5, no. 3 & 4, year 2012, <http://www.ariajournals.org/software/>

The copyright for each included paper belongs to the authors. Republishing of same material, by authors or persons or organizations, is not allowed. Reprint rights can be granted by IARIA or by the authors, and must include proper reference.

Reference to an article in the journal is as follows:

<Author list>, "<Article title>"
International Journal on Advances in Software, issn 1942-2628
vol. 5, no. 3 & 4, year 2012,<start page>:<end page> , <http://www.ariajournals.org/software/>

IARIA journals are made available for free, proving the appropriate references are made when their content is used.

Sponsored by IARIA

www.aria.org

Copyright © 2012 IARIA

Editor-in-Chief

Luigi Lavazza, Università dell'Insubria - Varese, Italy

Editorial Advisory Board

Hermann Kaindl, TU-Wien, Austria

Herwig Mannaert, University of Antwerp, Belgium

Editorial Board

Witold Abramowicz, The Poznan University of Economics, Poland

Abdelkader Adla, University of Oran, Algeria

Syed Nadeem Ahsan, Technical University Graz, Austria / Iqra University, Pakistan

Marc Aiguier, École Centrale Paris, France

Rajendra Akerkar, Western Norway Research Institute, Norway

Zaher Al Aghbari, University of Sharjah, UAE

Riccardo Albertoni, Istituto per la Matematica Applicata e Tecnologie Informatiche "Enrico Magenes" Consiglio Nazionale delle Ricerche, (IMATI-CNR), Italy / Universidad Politécnica de Madrid, Spain

Ahmed Al-Moayed, Hochschule Furtwangen University, Germany

Giner Alor Hernández, Instituto Tecnológico de Orizaba, México

Zakarya Alzamil, King Saud University, Saudi Arabia

Frederic Amblard, IRIT - Université Toulouse 1, France

Vincenzo Ambriola, Università di Pisa, Italy

Renato Amorim, University of London, UK

Andreas S. Andreou, Cyprus University of Technology - Limassol, Cyprus

Annalisa Appice, Università degli Studi di Bari Aldo Moro, Italy

Philip Azariadis, University of the Aegean, Greece

Thierry Badard, Université Laval, Canada

Muneera Bano, International Islamic University - Islamabad, Pakistan

Fabian Barbato, Technology University ORT, Montevideo, Uruguay

Barbara Rita Barricelli, Università degli Studi di Milano, Italy

Peter Baumann, Jacobs University Bremen / Rasdaman GmbH Bremen, Germany

Gabriele Bavota, University of Salerno, Italy

Grigorios N. Beligiannis, University of Western Greece, Greece

Noureddine Belkhatir, University of Grenoble, France

Imen Ben Lahmar, Institut Telecom SudParis, France

Jorge Bernardino, ISEC - Institute Polytechnic of Coimbra, Portugal

Rudolf Berrendorf, Bonn-Rhein-Sieg University of Applied Sciences - Sankt Augustin, Germany

Ateet Bhalla, Oriental Institute of Science & Technology, Bhopal, India

Ling Bian, University at Buffalo, USA

Kenneth Duncan Boness, University of Reading, England

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 Autónoma de México, México
Rafael del Vado Vírveda, 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, Indiana 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, Süleyman Demirel 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 University, 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, TU Wien, 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 Paraíba, 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
Constantinos 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 Nguyễn, INRIA Grenoble 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
Michael North, Argonne National Laboratory, USA
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
Eric Pardede, La Trobe University, Australia
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 Pustulka-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 Autònoma 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, Austrian Institute of Technology, Austria
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
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
Rainer Weinreich, Johannes Kepler University Linz, Austria
Stefan Wesarg, Fraunhofer IGD, Germany
Sebastian Wiczorek, 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

CONTENTS

pages: 146 - 165

A Bioinspired Coordination Strategy for Controlling of Multiple Robots in Surveillance Tasks

Rodrigo Calvo, University of Sao Paulo, Brazil
Janderson Rodrigo de Oliveira, University of Sao Paulo, Brazil
Mauricio Figueiredo, Federal University of Sao Carlos, Brazil
Roseli Aparecida Francelin Romero, University of Sao Paulo, Brazil

pages: 166 - 178

Quality Attributes for Web Services: A Model-based Approach for Policy Creation

Alexander Wahl, Department of Computer Science Furtwangen University of Applied Science Furtwangen, Germany
Bernhard Hollunder, Department of Computer Science Furtwangen University of Applied Science Furtwangen, Germany
Varun Sud, Department of Computer Science Furtwangen University of Applied Science Furtwangen, Germany
Ahmed Al-Moayed, BI/ HANA Department, Adweko GmbH Walldorf, Germany

pages: 179 - 190

Testing Object-Oriented Code Through a Specifications-Based Mutation Engine

Pantelis Stylianos Yiasemis, Cyprus University of Technology, Cyprus
Andreas Andreou, Cyprus University of Technology, Cyprus

pages: 191 - 199

Benchmarking Data as a basis for Choosing a Business Software Systems Development and Enhancement Project Variant – Case Study

Beata Czarnacka-Chrobot, Warsaw School of Economics, Poland

pages: 200 - 211

Mining Test Cases: Optimization Possibilities

Edith Werner, Neumüller Ingenieurbüro GmbH, Germany
Jens Grabowski, Institute for Computer Science, University of Göttingen, Germany

pages: 212 - 223

Synthesizing Control Software from Boolean Relations

Federico Mari, Sapienza University of Rome, Italy
Igor Melatti, Sapienza University of Rome, Italy
Ivano Salvo, Sapienza University of Rome, Italy
Enrico Tronci, Sapienza University of Rome, Italy

pages: 224 - 236

Dynamic Reverse Engineering of Graphical User Interfaces

Inês Coimbra Morgado, FEUP, Portugal
Ana C. R. Paiva, FEUP, Portugal
João Pascoal Faria, FEUP, INESC TEC, Portugal

pages: 237 - 251

Event-Sequence Testing using Answer-Set Programming

Martin Brain, University of Oxford, UK

Esra Erdem, Sabanci University, Turkey

Katsumi Inoue, National Institute of Informatics, Japan

Johannes Oetsch, Vienna University of Technology, Austria

Jörg Pührer, Vienna University of Technology, Austria

Hans Tompits, Vienna University of Technology, Austria

Cemal Yilmaz, Sabanci University, Turkey

pages: 252 - 262

Soft Constraints in Feature Models: An Experimental Assessment

Jorge Barreiros, Instituto Superior de Engenharia de Coimbra, Portugal

Ana Moreira, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal

pages: 263 - 277

Using Functional Complexity Measures in Software Development Effort Estimation

Luigi Lavazza, Università degli Studi dell'Insubria, Italy

Gabriela Robiolo, Universidad Austral, Argentina

pages: 278 - 292

Metrics and Measurements in Global Software Development

Maarit Tihinen, VTT Technical Research Centre of Finland, Finland

Päivi Parviainen, VTT Technical Research Centre of Finland, Finland

Rob Kommeren, Philips, The Netherlands

Jim Rotherham, Symbio, Finland

pages: 293 - 307

Quality-Oriented Design of Software Services in Geographical Information Systems

Michael Gebhart, Gebhart Quality Analysis (QA) 82, Germany

Suad Sejdovic, Campana & Schott, Germany

pages: 308 - 322

An Agile Driven Architecture Modernization to a Model-Driven Development Solution - An industrial experience report

Mina Boström Nakićenović, SunGard Front Arena, Sweden

pages: 323 - 334

Factors Leading to the Success and Sustainability of Software Process Improvement Efforts

Natalja Nikitina, KTH Royal Institute of Technology, Sweden

Mira Kajko-Mattsson, KTH Royal Institute of Technology, Sweden

pages: 335 - 344

Evaluating Performance of Android Systems as a Platform for Augmented Reality Applications

Andrés L. Sarmiento, University of A Coruña, Spain

Margarita Amor, University of A Coruña, Spain

Emilio J. Padrón, University of A Coruña, Spain

Carlos V. Regueiro, University of A Coruña, Spain

Raquel Concheiro, University of A Coruña, Spain

Pablo Quintía, University of A Coruña, Spain

pages: 345 - 357

Debugging Ubiquitous Computing Applications With the Interaction Analyzer

Nam Nguyen, UCLA, USA

Leonard Kleinrock, UCLA, USA

Peter Reiher, UCLA, USA

pages: 358 - 367

Restoration of Blurred Images Using Revised Bayesian-Based Iterative Method

Sigeru Omatu, Osaka Institute of Technology, Japan

Hideo Araki, Osaka Institute of Technology, Japan

Yuka Nagashima, Osaka Prefecture University, Japan

pages: 368 - 377

Energy-aware MPSoC for Real-time Applications with Space-Sharing, Adaptive and Selective Clocking and Software-first Design

Stefan Aust, Clausthal University of Technology, Germany

Harald Richter, Clausthal University of Technology, Germany

pages: 378 - 388

Footprint-Based Generalization of 3D Building Groups at Medium Level of Detail for Multi-Scale Urban Visualization

Shuang He, L'UNAM Université, Ecole Centrale Nantes, CERMA, France

Guillaume Moreau, L'UNAM Université, Ecole Centrale Nantes, CERMA, France

Jean-Yves Martin, L'UNAM Université, Ecole Centrale Nantes, CERMA, France

pages: 389 - 400

Subjective Assessment of Data Quality considering their Interdependencies and Relevance according to the Type of Information Systems

Maria del Pilar Angeles, Universidad Nacional Autónoma de México, Facultad de Ingeniería., México

Francisco Javier García-Ugalde, Universidad Nacional Autónoma de México, Facultad de Ingeniería., México

pages: 401 - 413

Analyzing 3D Complex Urban Environments Using a Unified Visibility Algorithm

Oren Gal, Technion - Israel Institute of Technology, Israel

Yerach Doytsher, Technion - Israel Institute of Technology, Israel

A Bioinspired Coordination Strategy for Controlling of Multiple Robots in Surveillance Tasks

Rodrigo Calvo, Janderson R. de Oliveira, and Roseli A. F. Romero

Department of Computer Sciences

University of Sao Paulo

Sao Carlos - SP, Brazil

Email: {rcalvo,jrodrigo,rafrance}@icmc.usp.br

Mauricio Figueiredo

Department of Computer Sciences

Federal University of Sao Carlos

Sao Carlos - SP, Brazil

Email: mauricio@dc.ufscar.br

Abstract—There are tasks that the multiple agent system approach is very appropriate for improving performance. Among them are: environment exploration, mineral mining, mine sweeping, surveillance, and rescue operations. The expected advantage is not a mere consequence of putting together many agents. An efficient coordination strategy is decisive to reach performance improvements. In the present paper, a new strategy is proposed for coordination of multiple robot systems applied to exploration and surveillance tasks. The coordination strategy is distributed and on-line. It is inspired in biological mechanisms that define the social organization of swarm systems; specifically, it is based on a modified version of usual artificial ant systems. Two versions of the proposal are evaluated. The experiments consider two performance criteria: the average of the numbers of surveillance epochs and average of the surveillance time intervals. Simulation results confirm that exploration and surveillance emerge from a synergy of individual robot behaviors. Data analyses show the coordination strategy is effective and suitable to execute exploration and surveillance tasks.

Keywords-multiple robot system; surveillance task; coordination strategy; ant colony system; swarm systems.

I. INTRODUCTION

A multiple agent system is well characterized if its dynamics reflect some synergy, that is, global behaviors emerge from the individual ones improving capabilities and performance to reach a specific goal. If only one agent of a group achieves equally the same goal with the same performance the entire group does, then at first, the group of agents are not a multiple agent system. Regarding as multiple robots system, this present paper is an extension of approach proposed in [1], where a robots team is able to monitor an environment independently of adopted configuration. In the other words, the way by which walls (or obstacles) are placed in an environment does not limit the accomplishment of exploration and surveillance tasks.

There are many applications to which multiple agent systems are the suitable approach to be adopted, such as: rescue operations in catastrophic events; fire extinction; and exploration in hostile environment [2][3][4]. Some of the main reasons that justify this choice, among others, are: great dimension of the task and reduced resources (e.g., velocity,

strength, energy) provided for a single agent; necessity to adaptation to spatial or temporal variation of service demands and robustness. For some tasks this approach is mandatory; for others it is a matter of convenience to increase the quality, to improve the performance or to save monetary funds.

Nowadays, the technology reaches more sophisticated levels providing environment support and embedded supplies. These improvements bring closer the possibility of multiple agent systems to become usual. The strong expectation associated to this possibility captivates the attention of the scientific community. Different aspects are investigated in multiple agent systems, such as: agent communication and information merging [5][6][7][8]. Another important aspect is the agent coordination that allows the system accomplishes efficiently general tasks such as: exploration, coverage, surveillance, among others.

On the one hand, coordination strategies are designed to provide multiple agent systems with a set of characteristics, e.g., decentralized coordination, small redundancy of agent efforts and strong cooperative behavior. On the other hand, designers devote effort to propose coordination strategies that are dependent on the least number of parameters as possible. A tricky parameter is the number of agents. Another requirement that may depreciate the strategy is the need to have total knowledge of the environment.

According to a technique described in [9] robots construct a common map cooperatively. It is introduced the notion of a frontier, which is a boundary between the explored and unexplored areas. As robots move, new boundaries are detected and frontiers are grouped in regions. Then, the robots navigate toward the centroid of the closest region, while sharing maps. The strategy is a centralized type since A* algorithm considers all information that the robots provide and the algorithm output defines the next steering direction of each robot. The strategy does not avoid unnecessary redundancy of robot efforts.

The problem of surveillance using multiple agents is investigated as a problem of a cooperative patrolling in [10]. A mathematical formulation is proposed as a minimization

problem. The objective to minimize the refresh time, that is, the time necessary for the agents patrol completely the environment. The solution they find is an approximation algorithm of polynomial computational cost based on a topological graph representation and a path-covering procedure. The strategy depends totally on the knowledge of the environment.

Methods based on stimergy fields for cooperation have been recently employed in the context of robotic exploration [11][12][13]. They rely on a mechanism of indirect communication among the agents which allows their actions to be influenced by a trace left previously in the environment by the robot. In this way, a task can be accomplished in an efficient manner. A different coordination scheme in [14] is proposed based on potential fields in which repulsive forces repel robots from each other and obstacles. Starting navigation from the same region, the robots keep moving until repulsive forces cancel each other. At this moment the sensor network is settled and robots stop. This approach ensures the coverage task if the number of robots is sufficiently great. Unfortunately, the authors do not show how to find the minimum number of robots. Therefore there always exists a possibility that the strategy fails.

Coverage tasks are the focus of the investigation in [15]. The distributed coordination strategy, based on the Voronoi diagram and Delaunay triangulation, is proposed to maximize the connected coverage area. The strategy is robust to robot failures. Voronoi diagram is also adopted to solve the connected coverage problem in [16]. Despite these strategies solve a connected coverage problem, both do not sense completely the environment, that is, not all parts of the environment are visited by any robot.

The problem of coordination of multiple agents is considered complex [17][6]. Coordination strategies based solely on mathematical formulation and on agent and environment models are very parameter dependent and suffer critical degradation due to agent failure [18][15][16]. Furthermore, the problem of coordination of multiple robots that execute a surveillance task is proved to be NP-hard [10]. Bio-inspired theories provide fundamentals to design alternative strategies that overcome the main difficulties that become traditional strategies vain [19][20].

Particularly, the artificial analog versions of biological mechanisms that define the social organization dynamics, observed in some swarm systems, are very appropriate in applications involving multiple agents, for example, decentralized control, communication and coordination [21][22][23].

In our previous works, some initial ideas about construction of a new bioinspired based model for a control strategy of multiple robots were proposed in [24]. It is named *Inverse Ant System-Based Surveillance System* (IAS-SS). In a preliminary model of IAS-SS, it was considered distinct steering direction mechanisms and the feature of robustness in regarding the number of robots adopted. As an

extension, in [1] was shown that the system does not depend on knowledge of the environment, where the agents act in environments with different configurations (arrangement of obstacles). In order to prove the efficiency of communication way among the agents rather than that adopted by biological agents, a parametric analysis was performed in [25], using various stigmergy mechanisms.

In the present work, an enhancement of IAS-SS strategy is proposed through a more complete description. It is designed according to a modified version of the ant system algorithm presented in [26]. In this strategy, the agents were able to indirect communication as the biological agents are, but their reaction to the pheromone is distinct, steering directions are defined to guide preferably the robot to where there is low quantity of pheromone. IAS-SS strategy is primarily for the coordination of multiple robots applied to surveillance and exploration tasks. Some characteristics of IAS-SS are: decentralized, on-line, and parameter independent from both the number of robots and the environment structure. Two versions of the strategy IAS-SS are compared with a total random strategy. Different experiments are considered, each of which varying a specific parameter: number of robots, the environment scale, and initial position. Results show that exploration and surveillance tasks are effectively executed and the respective general behaviors emerge from the individual robot behavior (move to where there is less pheromone).

Since the task of modeling all possible events, accurately, in real world through mathematical models is not trivial, the main contribution of this paper is a simple coordination strategy based on a modified version of the traditional ant system, that is, the robots are attracted to the region of the environment with low amount of pheromone. The behavior of exploration and surveillance are generated only by the information supplied by the deposited pheromone with few parameters to be adjusted. It is necessary neither robot's position nor their local map environment. It is worth to be emphasized the way in which the robots deposit pheromone. This substance is left in the frontal area of robots, instead of the positions occupied that generate a pheromone trail. These characteristics are not found in other approaches existing in the literature.

This paper is organized such as it follows. In Section II is presented the basic concepts of the artificial ant system theory. In Section III, the mathematical formulation of the surveillance problem is presented. The multiple robot system and the coordination strategy IAS-SS are focused in Section IV. The pheromone evaporation dynamics, the mechanisms of pheromone releasing, and the procedure to determine the robot steering direction are also defined. In Section V simulation results are reported. The main contributions and relevant aspects of this paper as well as expectations for future works are highlighted in Section VI.

II. ANT SYSTEM

Surprisingly, the complex tasks that ant colonies perform, such as object transportation and build edges, demand relatively more capabilities than a single ant is endowed [27][28].

Biological ants have two known mechanisms to establish communication, namely, direct and indirect. Biological ants not only exchange stimuli when they meet; but also exchange stimuli indirectly (a communication mechanism called stimergy). Ants deposit a specific type of substance (pheromone) on the ground while they move. There are different types of pheromone, each of which associated with a particular meaning. If a pheromone trail is found and this pheromone type indicates food, then more and more ants follow this trail, depositing more pheromone and reinforcing the stimuli. An opposite behavior happens if the pheromone is of the aversive type, indicating risk and danger. Stimergy mechanism is considered as one of the factors that decisively contribute to amplify the capabilities of a single ant. Ant colonies use the stimergy mechanism to coordinate their activities in a distributed way [29].

Artificial ant systems are the artificial counterparts of the biological ant colonies, designed to solve complex problems, among others: optimization combinatorial problems [26]. Analogously artificial ants (e.g., robots) are able to use the stimergy communication. Pheromone trail provides a type of distributed information that artificial agents may use to take decisions or modify to express previous experiences [30]. A distributed coordination behavior emerges from this capability, providing solutions to problems associated with exploration in hyper-spaces.

III. DEFINITIONS AND PRELIMINARY CONCEPTS

There are different mathematical formulations in the literature. For example, in [10] the concept of *viewpoint* is defined. *Viewpoints* are specific points in the environment such that from those points it is possible to sense the whole environment. Then the robots have to go to them repeatedly in order to keep the environment sensed completely. The optimal surveillance task is defined as the minimization of the largest interval between two consecutive instants that any robot reaches a particular *viewpoint*, considering all *viewpoints* and during all time the task lasts.

Informally, the surveillance task means the task of keeping endlessly a target under closed observation. In this paper, the target is an environment. It is not necessary to keep all points of the environment under observation at the same time, but every point has to be observed repeatedly while the task lasts. If a set of agents are considered to carry out the task, the agents have to follow trajectories that allow them to sense all parts of the environment again and again. Clearly, it is not necessary that each agent goes to every point, but every point has to be observed by at least one agent (anyone) repeatedly. Then, the execution of the surveillance task is considered effective if the environment is completely and

continually sensed. Moreover, the smaller is the maximum interval between two consecutive sensing, considering any particular point of the environment, the more efficient is the execution of the surveillance task.

Two terms used in this work help the readers to understand how the surveillance task is evaluated in this work: *Surveillance Epoch (SE)* and *Surveillance Interval (SI)*. A surveillance interval is any interval of time in which all points of the environment are sensed at least once; and at least one point is sensed exactly once. This interval corresponds to a portion of the surveillance task and this portion is called surveillance epoch. If the surveillance intervals are considered from the start of the task, the SI's are uniquely determined. Starting from T_0^* , the agents start to move through the environment, sensing the environment. After a time, precisely at time T_1^* , all points are sensed at least once; and at least one point is sensed exactly once. The interval between T_0^* and T_1^* is the first SI. The second SI begins at T_1^* . It is important to notice that at T_1^* no point is considered sensed anymore, that is, a new reckoning starts at T_1^* to indicate the sensed points. The agents keep moving continuously. At T_2^* all points are sensed at least once; and at least one point is sensed exactly once (considering a new reckoning starts for the second SI). The interval between T_1^* and T_2^* is the second SI. All other SI's are defined analogously. The surveillance task is evaluated measuring the maximum length of the intervals SI.

In order to put the meaning of the surveillance task more rigorous the respective mathematical model is built next.

Consider that robots r_k , $k = 1, \dots, N$ move in a planar space $Q \subset \mathbb{R}^2$ and that an arbitrary point in Q is denoted by q . Assume that the time t is discrete. Let $L_t^k, L_t^k \subset Q$, be the area that the r_k -th robot senses at instant t . Hence, the r_k -th robot senses a point q at instant t , if $q \in L_t^k$. Define a function $I_k(\cdot, \cdot)$ to associate a point q with the respective state considering the r_k -th robot, that is, $q \in L_t^k$ if and only if $I_k(q, t) = 1$. Define also a function $\Omega_i(\cdot, \cdot)$ to associate a point q with the number of times it is sensed from T_{i-1}^* up to t , considering all robots r_k , that is:

$$\Omega_i(q, t) = \begin{cases} 0, & \text{if } t = T_{i-1}^* \\ \sum_{\mu=T_{i-1}^*}^t \sum_{k=1}^N I_k(q, \mu), & \text{otherwise} \end{cases} \quad (1)$$

Then, this paper focuses on the minimization problem such as it follows:

$$\min_{1 \leq i \leq C} \max \left(T_i^* - T_{i-1}^* \right) \quad (2)$$

subject to:

$$\begin{aligned} \Omega_i(q, T_i^*) &\geq 1, \quad \forall q \in Q \\ \exists q \in Q | \Omega_i(q, T_i^*) &= 1, \end{aligned}$$

$$\sum_{i=1}^C (T_i^* - T_{i-1}^*) \leq T^F$$

where: T_i^* and T_{i-1}^* are the limits of the i -th surveillance interval; $C \geq 0$ is number of completed SE and T^F denotes the time when the surveillance task ends.

It is important for the reader to notice that the problem defined earlier in equation 2 is not solved here exactly, but only in an approximated way. This is almost a rule since this problem is known to be very complex, when the number of robot is big. This is the case in this paper. Multiple identical mobile robots are considered to carry out the surveillance task. Every robot is equipped with a set of sensor devices that allow the robots observe the environment. According to [10] the surveillance task problem such as described there is a NP-hard problem.

One among the aspects of the surveillance task problem considered in this paper is that the environment is unknown. By focusing on this aspect, it is important to notice that the exploration task may be regard as part of the surveillance task, since the environment is completely explored at the end of the first surveillance epoch. At this time all points of the environment are sensed, that is, there is no point remains to be found. Then, in what follows the focus is on the surveillance task and the exploration task is a mere consequence.

IV. INVERSE ANT SYSTEM-BASED SURVEILLANCE SYSTEM (IAS-SS)

The multiple agent system approach is adopted to solve the surveillance task problem such as described in the former section. The agents are multiple identical mobile robots each of which equipped with a sensor for detecting a particular characteristic of the environment.

At first glance, this approach seems attractive, that is, a plausible conclusion is: multiple robots execute more efficiently the surveillance task than a single one. However this conclusion is true if, at least, there is a capable strategy to coordinate suitably the robots. It means that the strategy has to generate trajectories for every robot, leading them repeatedly to all parts of the environment, satisfying some performance criteria, e.g., minimization of the interval between two consecutive instants a point in the environment is sensed, considering all points.

Putting together: performance requirements, solution restrictions, and strategy characteristics; makes the design work a hard task (such as asserted in [10], see comments in the previous section).

The system proposed, called *Inverse Ant System-Based Surveillance System* (IAS-SS), is designed according to the main ideas of the artificial ant system. In short, the IAS-SS system is a multiple robot system. The robot's cybernetic system consists of two components: the navigation controller and the pheromone disperser. The pheromone disperser is

one of the components of the robot's cybernetic system, since it is related to the robot's indirect communication system. The Figure 1 represents the components of the cybernetic system and the respective connections with other elements of the system, including the environment.

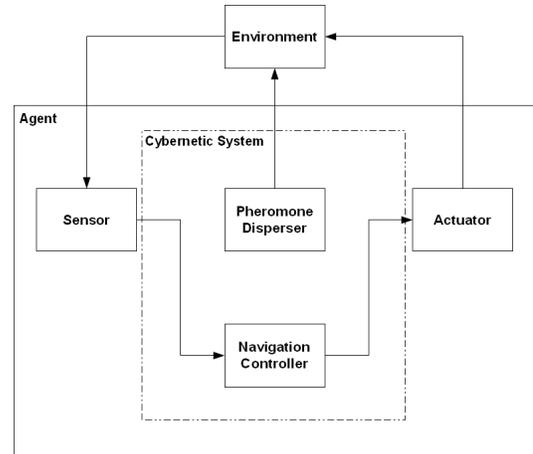


Figure 1. Cybernetic system architectural diagram for a single robot.

The strategy, called *IAS-SS strategy*, is for coordination of the IAS-SS system's robots applied to surveillance tasks. The coordination strategy is a distributed one, that is, every robot moves independently and takes decisions based on the stimuli it receives from the environment. The IAS-SS coordination strategy is a reactive (real-time) strategy, does not generate decision dead-locks, and is computational low-cost.

The IAS-SS coordination strategy is based on the indirect communication mechanism (*stigmergy*) the biological ant colonies exhibit. The IAS-SS coordination strategy generates the following general system dynamics. While the robots navigate they deposit a specific substance into the environment. This substance is called *pheromone*, since it is the analogue of the pheromone in biological ant colonies. At each time the robot sensor detects the stimuli from the environment corresponding to the total amount of the pheromone deposited on the area defined by the sensor range. The amount of pheromone detected is the accumulated pheromone deposited on that area considering all robots. After that the robot adjusts its navigation direction, deposits the pheromone and moves.

The IAS-SS strategy is completely described in the next subsections. Among other elements described, are: the navigation system and the steering direction mechanisms; and pheromone disperser. All other elements of the IAS-SS system will be considered as well. In the end, all of them will be described in detail.

A. Navigation System

According to the IAS-SS coordination strategy, the robot navigation system consists of two subsystems: surveillance

system and obstacle avoidance system. Only one is active at each time. Most part of the time the surveillance system is active and the steering direction is determined according to it. The trajectories the surveillance navigation system generates cause the robots execute the surveillance task. The trajectories do not lead robots to collision situations as well, with rare exceptions. In order to avoid completely any possible collision situation, the obstacle avoidance system is active only if the robot is very close to a wall or another robot.

Although the obstacle avoidance system is not the main concern in this paper, it is briefly described. As long as robots are close to an obstacle, the amount of pheromone in its boundary region is increased. Then, the robots generate the obstacle avoidance behavior due to the high amount of pheromone. Therefore, obstacle avoidance is an emergent behavior of the IAS-SS strategy. The trajectories generated by the strategy does not guide the robots to a collision situation. Besides the exploration and surveillance tasks, the robots are able to avoid obstacles, keeping a reasonable distance from them.

However, there are some exceptions when a robot collides against another robot or against an obstacle according to its physical characteristics. In this sense, it is used a mechanism for obstacle avoidance based on fuzzy logic [31] that adjusts the steering direction of the robot using the information about its distance to an obstacle. The details of the mechanism based on fuzzy logic can be found in [32]. It is enough to say that this mechanism is active only when the distance between the robot and an obstacle is smaller than a predefined constant η .

The general description of steering direction mechanism the surveillance navigation system implements is such as follows. At each time a set of stimuli is detected, corresponding to the amount of the pheromone deposited at different angles and same specific distance (at the range border) in front of the robot. The lesser is the detected amount of the pheromone detected the greater is the probability that the robot takes the navigation direction equal to the angle where this amount of pheromone is.

According to this strategy robots tend to move to the directions where there is low amount of pheromone. The general robot behavior observed is that the robot moves to unexplored areas or areas robots seldom visit.

Considering the IAS-SS coordination, the logic associated with the decision that chooses the steering direction angle is opposite of that adopted in the traditional ant system theory. The logic adopted there generates a positive feedback, that is, the greater is the amount of the pheromone the greater is the probability of the agent to follow the respective direction.

Two versions of the steering angle mechanism are described. The first one, called Stochastic Sampling, considers all possible pheromone stimuli that the sensor detects at the border of its range. The second, called Best Ranked

Stochastic Sampling, determines the adjusting of steering angle based on a select set of stimuli detected at the border of the sensor range.

The mathematical models for these two versions of the steering angle mechanism are described as it follows. Before that, consider two assumptions. First, there are N identical mobile robots r_k , $k = 1, \dots, N$. Second, the model of the sensor adopted is such that it detects pheromone stimuli at the border of its range (Figure 2). The border is a circumference of a circle of radius R , ahead of the robot, from 90 degrees to the left to 90 degrees to the right of the steering direction. The total range of 180 degrees is divided in identical angle intervals each of which measuring α degrees. The middles of the intervals are settled on angles A_s , such that: $(2S + 1)\alpha = 180$ and $A_s = s\alpha$, where $s \in [-S, S]$ and $s \in \mathbb{N}$.

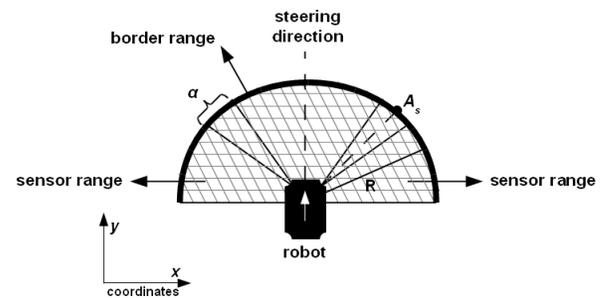


Figure 2. Robot and sensor models

1) *Stochastic Sampling Mechanism*: A pheromone stimulus corresponds to the amount of pheromone deposited in an angle interval. A probability value assigned to each discrete angle A_s is inversely proportional to the amount of pheromone deposited in the angle interval that is settled on the angle A_s . The lower is the amount of pheromone detected in the angle interval, the higher is the probability associated with the respective angle A_s . Specifically, the probability $P(s)$ assigned to the angle A_s is:

$$P(s) = \frac{1 - \tau_s}{\sum_{i=-S}^S (1 - \tau_i)} \quad (3)$$

where τ_s is the amount of pheromone corresponding to the angle interval A_s .

The adjustment of the steering direction is determined according to a discrete random variable a defined through the probability $P(s)$, assuming values in the set $\{A_s \mid s = -S, \dots, -1, 0, 1, \dots, S\}$.

At each time t , the adjustment of steering direction is given by:

$$\Theta_k(t) = \Theta_k(t-1) + \gamma A_s^* \quad (4)$$

where: $\Theta_k(t)$ is the steering angle of the robot k at instant t , $\gamma \in [0, 1]$ is the constant coefficient for smoothing the steering direction adjustment; and $A(s^*)$ is value of the random variable a at instant t for some $s = s^*$.

However, Stochastic Sampling mechanism is not efficient for large areas where the amount of the pheromone deposited is similar on every point. In this case, the amount of pheromone differs a bit and the A_s^* chosen may define bad steering directions due to the stochastic nature of the mechanism. For reducing the possibility of this shortcoming, a second different mechanism is described and investigated below.

2) *Best Ranked Stochastic Sampling Mechanism*: Differently from the Stochastic Sampling Mechanism, not all angle A_s are considered to define the steering direction, but only two subsets, U and V , such that the respective cardinalities are φ and ω ; and $\varphi + \omega \leq (2S + 1)$. The subset U consists of angles A_s associated with the least detected amount of pheromone. The subset V consists of elements chosen randomly, according to an uniform distribution, from the angles A_s that are not in the subset U .

The rules for building the subsets U and V are such as follows:

- **Subset U**

if $A_s \in U$ and $A_z \notin U$, then $\tau_s \leq \tau_z$

- **Subset V**

if $A_s \in V$, then $A_s \notin U$; and A_s are chosen randomly

where: τ_s is defined according to equation 3 and $s, z = -S, \dots, -1, 0, 1, \dots, S$.

A probability value is assigned to each discrete angle in both of the subsets U and V . The probability assigned to the angle A_s is inversely proportional to the amount of the pheromone deposited in the respective angle interval and it is defined such as:

$$\bar{P}(s) = \frac{1 - \tau_s}{\sum_{i \in \{s | A_s \in (U \cup V)\}} (1 - \tau_i)} \quad (5)$$

Consider $A_s = A^*$, A_s chosen according to a discrete random variable a defined through the probability $\bar{P}(s)$, assuming values in the set $\{A_s | A_s \in (U \cup V)\}$. At each time t , the adjustment of steering direction is given by equation 4:

The basic steps of Best Ranked Stochastic Sampling are described in the Algorithm 1 for a single robot.

B. Pheromone Releasing and Evaporation

In traditional artificial ant systems, agents release pheromone on the ground only on their respective positions signaling exactly the robot way [26]. Differently, in this

Algorithm 1 The Best Ranked Stochastic Sampling Algorithm

- 1: Initialize the parameters φ and ω
 - 2: Detect the amount of the pheromone in the border of the sensor range
 - 3: Build the subsets U and V
 - 4: **for** every angle interval $A_s \in (U \cup V)$ **do**
 - 5: Assign to A_s the probability $P(s)$ according to equation 5
 - 6: **end for**
 - 7: Define the next steering direction of the robot according to equation 4
 - 8: Back to step 2
-

article, the artificial agents in IAS-SS spread out pheromone on a wide area in front of their respective positions, corresponding to the sensor range area.

After the agent determines the steering direction (see equation 4), but before it moves to, it spreads pheromone. The amount of the pheromone deposited on the ground decreases as the distance from the robot increases. Consider that L_t^k is the sensor range area of the k th robot at the iteration t and Q is the entire environment space, respectively, such that $L_t^k \subset Q \subset \mathbb{R}^2$. Then, the amount of the pheromone $\Delta_q^k(t)$ the k th robot deposits at the position q at iteration t is given by:

$$\Delta_q^k(t) = (\tau_{max} - \tau_q(t-1))\Gamma_q^k(t), \text{ and} \quad (6)$$

$$\Gamma_q^k(t) = \begin{cases} \delta e^{-\frac{(q-q_k)^2}{\sigma^2}}, & \text{if } q \in L_t^k \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

where: q_k is the position of the k th robot; τ_{max} is the maximum amount of pheromone; σ is the dispersion; and $\delta \in (0, 1)$.

Multiple robots deposit pheromone in the environment at same time, then the total amount of pheromone deposited on the position q at iteration t depends on the contribution of every robot.

Furthermore, pheromone is not a stable substance, that is, it evaporates according to a specific rate. The total amount of the pheromone that evaporates $\Phi_q(t)$ at position q and time t is modeled according to:

$$\Phi_q(t) = \rho\tau_q(t) \quad (8)$$

where: ρ is the evaporation rate; and $\tau_q(t)$ is the total amount of the pheromone on the position q at iteration t .

Therefore, the total amount of the pheromone $\tau_q(t)$ at q and at time t is given by:

$$\tau_q(t) = (\tau_q(t-1) - \Phi_q(t-1)) + \sum_{k=1}^N \Delta_q^k(t) \quad (9)$$

V. EXPERIMENTAL RESULTS

Experiment simulations are developed to evaluate preliminarily the bioinspired coordination strategy IAS-SS based on ant colony algorithm, whereas pheromone causes repulsive behavior of ants instead of attractive for surveillance task. The expectation for surveillance task consists of keeping robots moving among regions of the environment in order to patrol wholly it constantly. The strategy is considered to generate the dynamics of the multiple robot system applied to exploration and surveillance tasks.

The Player/Stage platform (<http://robotics.usc.edu/player>) is used to perform the experiments. The Player/Stage is a robot server designed by the University of Southern California for distributed control (www.usc.edu). Player operates in a client/server environment and the communication between them occurs through TCP/IP protocol. Stage is a simulator for robots and sensors for two-dimensional environments. Player/Stage models various robots and sensors simulating simultaneously their exact dynamics, including odometric error models. For the purpose of the experiments, the robot Pioneer 2DX is chosen to be modeled in the Player/Stage platform. This robot is equipped with a laser range-finder able to scan the environment (general obstacles, e.g., walls and objects).

The experiments are arranged in four groups. The first consists of experiments focusing on the steering direction mechanisms described in Section IV-A. The mechanisms are compared with a completely uniform one. The second group of experiments is designed to investigate the influence of the configuration of robot initial positions in the task performances. The experiments in the third group concerns specifically the robustness of the coordination strategy regarding to the environment structure. Finally, an analysis the impact of the number of robots on the system performance is presented in of fourth group.

Since the surveillance task requires the robots are in constant moving, the IAS-SS system have to experiment distinct challenges in face of different situations to develop navigation strategies. In order to maximize the watched area at the same instant, the goal of IAS-SS system is to keep robots in different regions avoiding the waste of sensor resource and reducing the time which an area is non-monitored. System efficiency is measured by how short is the time in which a region is non-monitored without sharing regions among robots.

The experimental data are selected and compiled assuming the following meaning. First, the exploration task is executed if the environment is completely covered, that is, the system is capable to provide information to map the environment completely. Moreover, the faster the system completes the task, the better is its performance. Second, the system carries out the surveillance task if there is no instant T^* such that after this instant exists a region that is

not sensed anymore. Despite this definition for surveillance task is accurate, it is not suitable since may be impossible to find T^* . Therefore, for practical purposes, it is important that the system concludes the task continually, that is, the system has to be able to sense the entire environment considering that a new sensing task is started when the system concludes the previous one. Furthermore, the lesser is the maximum time between two consecutive sensing tasks, the better is its performance.

The approach proposed to multi-robot coordination assumes that the environment is represented by occupancy-grid [33]. It uses a reticulated and probabilistic representation of information for modeling the unknown environment according to its laser range-finder readings. It is defined as a multidimensional random field that contains stochastic estimate of the cell states (occupied, not occupied or unknown) in the reticulated space. Each robot builds its own map as it moves and local maps are centralized resulting in a map of explored environment during surveillance task. Mapping module is independent of IAS-SS system. However, it is integrated to verify the explored area while the monitoring occurs. For releasing pheromone, IAS-SS strategy uses the same map generated and allocates in each cell reached by distance sensor a value that corresponds to amount of pheromone in this local.

The environments where IAS-SS system carries out tasks are divided in connected small regions called here rooms. In the context of this following experiments, a room is said to be visited if its central point is reached by any robot. In this case, the group of all central points corresponds to the set $Q \in \mathbb{R}^2$. Hence, the scenario considered here is an instance of the problem formulated in Section III. The system parameters adopted in the experiments are:

- Pheromone releasing and evaporation dynamics:
 - $\sigma = 0.43R$ (radius of the semicircle where the pheromone is deposited, see Figure 2);
 - $\rho = 0.01$ (evaporation rate); and
 - $\tau_q(0) = 0.5$ (the amount of pheromone at iteration $t = 0$).
- Robots and sensors:
 - $R = 8.00$ meters (radius of the semicircle where the pheromone is deposited, see Figure 2);
 - $\gamma = 0.5$ (constant coefficient for smoothing of steering direction adjusting); and
 - Robot speed: 0.5 meter per second.
- Steering direction mechanisms:
 - $S = 360$ (number of angle intervals).
- Simulation parameter:
 - $\eta = 0.3$ meter (maximum distance between the robot and an obstacle to trigger the obstacle avoidance system);
 - Time is discretized by simulation iterations: $t_s \in \mathbb{N}$;

- Maximum number of iterations = 1000.

These parameter values correspond to those that the multiple robot system reaches the best performance, considering all previous experiments executed. Due to aleatory characteristic of mechanisms for adjustment of steering direction, all experiments are executed 10 times (trials). Thus, average of performances are computed to evaluate them. The discrete time is adopted in simulation and it is equivalent to the number of iterations.

In the case of implementation of coordination strategy, all robots are in the same coordinates system. There is a global map of environment, modeled as occupancy grid, where each cell hosts a value that represents its state and the amount of pheromone in this respective local, indicating the time this cell was not monitored. Initially, all cells are setup as unknown with amount of pheromone as $0.5 (\tau_q(0))$. Since cells provides similar amount of pheromone, the decision make of robots tends to random characteristic. There, it takes some time to robots spread out. As long a robot moves, it builds its own local map in order to transfer it to the global map. Thus, a robot is able to detect pheromone left by other one, because all information about pheromone is available in global map. The position of robot is not relevant in this coordination strategy. The substance deposited by robots is enough to keep them far from each other.

During navigation, robots detect pheromone only in the cells that coincide to border of pheromone sensor. One of cells is elected through probability of equation 3 or 5 for SS and BRSS mechanisms, respectively. Then, the adjustment of steering direction towards the elected cell occurs by 4. Before moving, robots release pheromone (i.e., assign values to cells of occupancy grid) on all cells covered by range of distance sensor, according to equation 9.

A. Uniform versus Stochastic versus Best Ranked Stochastic Sampling

Both steering direction strategies, Stochastic Sampling (SS) and Best Ranked Stochastic Sampling (BRSS), have profound random characteristics, since the steering direction adjustment is determined according to a discrete random variable. In order to show that the respective performances are not a mere consequence of a random behavior, the strategies are compared with a uniform strategy (US). This strategy is able to execute neither the exploration nor the surveillance tasks. Different compiled data sets are considered to assess the strategies, namely: time to conclude the exploration task; and time interval between two consecutive sensing of any specific region. According to US strategy, a discrete random variable, defined by a uniform distribution in the space of the angles A_s , determines the steering direction adjustments. Observe that there is no connection between the pheromone and the uniform strategy; different from the SS and BRSS strategies.

The environment designed for evaluation is such as in Figure 3. It is possible to identify six rooms. Three robots k , $k \in \{1, 2, 3\}$, start the navigation at the room 1.

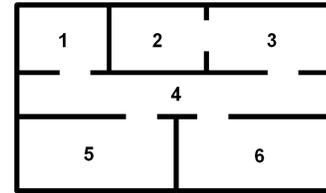


Figure 3. Environment structure

The performance of IAS-SS system according to mechanisms for adjustment of steering direction. Two aspects are considered for analysis: the time necessary to conclude the exploration task (SE); and maximum time interval between two consecutive sensing of any specific region (SI). The data are in Table I with the respective standard deviation for the average of number or SE and the average of SI considering 10 trials for each experiment. The performance of IAS-SS system is improved gradually as shown in graphic of Figure 4. It shows the boxplots of average of surveillance intervals of three mechanisms. The numbers 1, 2 and 3 refer to US, SS and BRSS mechanisms, respectively.

Table I
PERFORMANCE OF MECHANISMS FOR ADJUSTMENT OF STEERING DIRECTION

Mechanism	Average of Number of SE	Average of SI (iterations)
US	0.28 ± 0.5	796 ± 395.68
SS	4.28 ± 1.11	233.26 ± 82.43
BRSS	7.25 ± 1.71	122.13 ± 24.57

Additional information about the behavior of the system can be gathered observing the Figure 5. It exhibits three sets of graphics that summarize the simulation conducted, each of which corresponding to a different strategy. Data used to plot the graphics are from the trial with the median number of SI. For each strategy, three graphics are presented, each of which registering the behavior of one of the robots. The y-axis represents the rooms and the x-axis represents the iterations. Each vertical line indicates the SE, that is, the iteration when IAS-SS senses the whole environment (the robots visit cooperatively all the 6 rooms), considering that a new sense task is started after the system concludes the earlier one.

Considering the exploration task, the graphics show that the IAS-SS system with the Uniform Strategy is able to conclude the exploration task, but after a long time, precisely at the iteration 624. Observe that with strategies SS and BRSS the IAS-SS system executes more efficiently the task, that is, the system concludes the task very earlier, at the iterations 126 and 120, respectively. The IAS-SS system with US strategy concludes the surveillance task only once (there

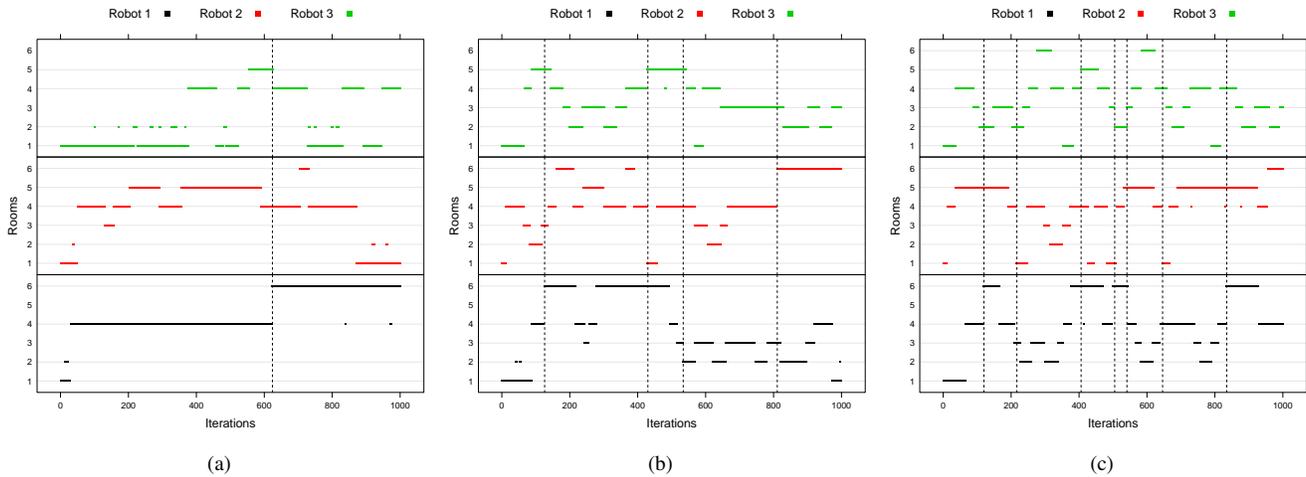


Figure 5. IAS-SS performance according to different strategies: (a) US; (b) SS; (c) BRSS mechanism.

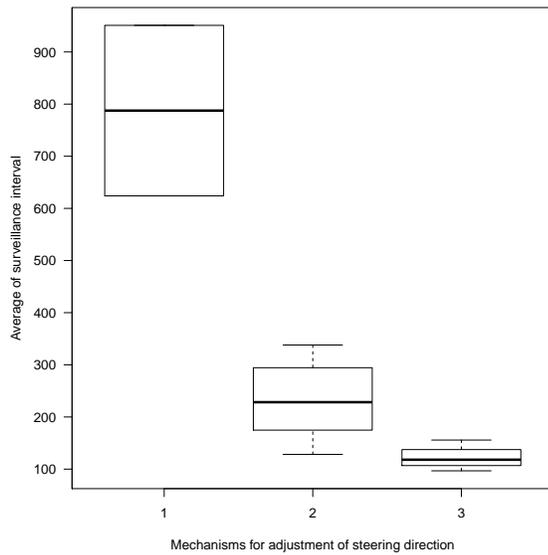


Figure 4. Boxplots of distribution of the average of surveillance intervals for different mechanisms for adjustment of steering direction

is only one SE), considering all the simulation. There is a strong contrast if this performance is compared with those obtained with the strategies SS and BRSS. Vertical lines indicate that the system with these strategies continually concludes the surveillance tasks (all the robots cooperatively visit the 6 rooms).

Pheromone distribution in an environment is an evidence of efficiency of strategy. The IAS-SS strategy is more efficient if the distribution of amount of pheromone is more egalitarian in entire environment. In this case, the average of amount of pheromone in all environment composes the map of pheromone distribution. It indicates the frequency when a specific region was visited (or monitored) in relative to

others. To comprehension of map, regions with low amount of pheromone are represented by blue color, whereas red color denotes regions with high amount of pheromone (or visitation frequency higher). Hence, to improve the best performance of strategy when BRSS mechanism is adopted, Figure 6 shows the average of amount of pheromone for mechanisms of adjustment of steering direction. According to the same data used to plot the graphics of Figure 5, the pheromone distribution is more uniform for BRSS strategy (Fig 6(c)). In contrast, it can be noted that more amount of pheromone in rooms 1 and 5 for execution of US strategy (Figure 6(a)).

These data are summarized in the Tables II and III. The system with SS strategy concludes the surveillance task 4 times and BRSS 7 times, and the maximum intervals between two consecutive conclusions are 304 and 189 iterations, respectively. The IAS-SS system with BRSS is clearly superior. The strategies SS and BRSS induce a stronger collaborative robot behavior than in the case of US strategy. Observe that robots in the pheromone dependent strategies vary more the rooms that they visit than robots do in the case of US strategy.

B. Initial position of robots

This group of experiments evaluates the efficiency of IAS-SS for distinct configuration of robots regards as their positions. Two cases of configuration of robots are designed in order to analyze the performance system: **1) together configuration** robots start navigation at same region (or room) and **2) separated configuration** at distinct rooms. For exploration and surveillance tasks, it is obvious the greater efficiency is guaranteed when the robots are not closer. However, experiments intend to demonstrate that, after a while, scenarios of joint configuration can achieve the same efficiency of separated configuration. Experiments to verify

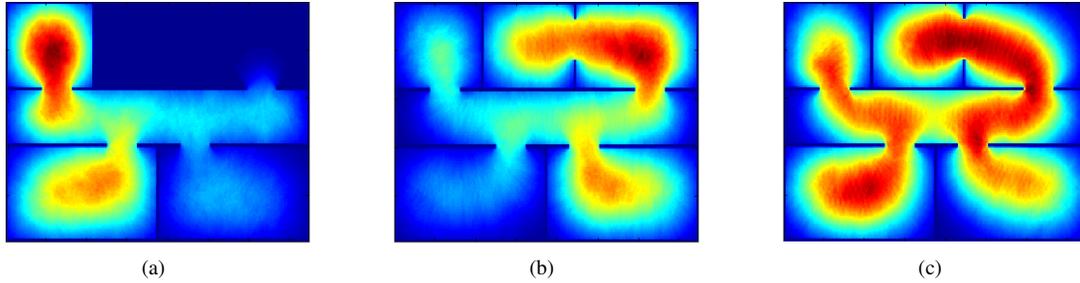


Figure 6. Maps of average of amount of pheromone according to strategies: (a) US; (b) SS; (c) BRSS.

Table II
SURVEILLANCE EPOCH FOR STEERING DIRECTION MECHANISMS

Mechanism	Max. SI (iterations)	Surveillance Epoch (iterations)						
		1 st	2 nd	3 rd	4 th	5 th	6 th	7 th
US	624	624	—	—	—	—	—	—
SS	304	126	304	104	277	—	—	—
BRSS	189	120	97	189	99	36	105	189

Table III
MONITORED ROOMS AT EACH SURVEILLANCE EPOCH

Mechanism	Robot	Monitored Rooms						
		1 st	2 nd	3 rd	4 th	5 th	6 th	7 th
US	# 1	1,2,4,6	—	—	—	—	—	—
	# 2	1,2,3,4,5	—	—	—	—	—	—
	# 3	1,2,4,5	—	—	—	—	—	—
SS	# 1	1,2,4,6	6,4,3	6,4,3,2	2,3	—	—	—
	# 2	1,2,3,4	3,4,6,5,1	1,4	4,3,2,6	—	—	—
	# 3	1,4,5	5,4,3,2	5,4	5,4,3,1	—	—	—
BRSS	# 1	1,4,6	6,4,3	3,2,4,6	6,4	6	6,4,3,2	4,3,2,6
	# 2	1,4,5	5,4,1	1,4,3,2	4,1	1,4,5	5,4,1	1,4,5
	# 3	1,4,3,2	2,3	2,3,4,6,1,5	5,4,3,2	2,3	3,4,6	4,3,2,1

the performance according to initial position of robots is accomplished in environment of Figure 7. For both of cases of configuration, three robots are launched. In particular for separated and together configurations, they start navigation at rooms 2, 6 and 7; and room 1, respectively.

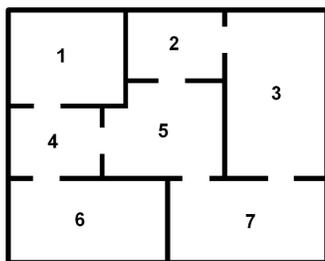


Figure 7. Environment structure

For the next experiments, six environment configurations are generated from combination of cases of configurations (separated and together) and steering direction mechanisms of experiments of Section V-A. Analogously to the previous experiments, two aspects are considered for analysis of the performance of IAS-SS system. As can be seen in the

Table IV, the separated configuration with SS mechanism yields the best performance, regarding both number of SE (nb. of SE) and average of SI (av. of SI). In the case of together configuration, the best results are obtained with the BRSS mechanism, which is, in fact, the best overall strategy. The Figure 8 shows the boxplots of the surveillance intervals of the six environment configurations. The numbers 1, . . . , 6 refer to environment separated configuration with US, SS and BRSS mechanism; and environment together configuration with US, SS and BRSS mechanisms, respectively.

The key of surveillance task is minimizing the time (iterations) which a region is non-monitored. Hence, here, a manner to measure the system performance is to analyze the maximum period (maximum number of iterations) which each room is non-visited. Maximum periods that rooms are non-visited are presented in Figure 9 for the six environment configurations. Data used to plot the graphics are from the trial with the median of average of number of SE and average of SI. Although separated configuration presents slightly advantage over together configuration, since at iteration $t = 0$, three robots monitor three different rooms, the performances of both configurations are similar. One of main characteristics of IAS-SS system is the skill of

Table IV
PERFORMANCE OF ENVIRONMENT CONFIGURATIONS WITH MECHANISMS FOR ADJUSTMENT OF STEERING DIRECTION

Configuration	Uniform Sampling	
	nb. of S.E.	av. of SI.
Separated	1 ± 0.94	358.55 ± 272.67
Together	0.43 ± 0.53	247 ± 313.77
Configuration	Stochastic Sampling	
	nb. of S.E.	av. of SI.
Separated	6.66 ± 1.73	160.04 ± 85.65
Together	6 ± 2.16	185.03 ± 122.4
Configuration	Best Ranked Stochastic Sampling	
	nb. of S.E.	av. of SI.
Separated	6 ± 2	199.71 ± 101.62
Together	7.85 ± 2.61	120.51 ± 44.3

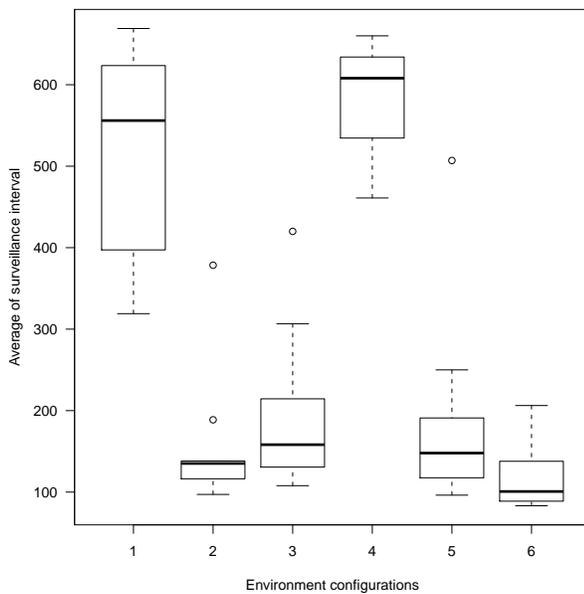


Figure 8. Boxplots of distribution of the average of surveillance intervals for the different adjustment of steering direction mechanisms

robots to keep distance from each other according to aversive pheromone. Then, even with together configuration, as long the robots move, they are spread in environment. Thus, the performance of together configuration becomes similar to separated configuration. That is, the advantage of separated configuration is diluted during navigation. To illustrate this scenery, graphics of Figure 10 show the behavior of robots and surveillance intervals for separated and together configurations using BRSS mechanism. They show that the IAS-SS system with the separated configuration concludes the SE task 6 times while the together configuration takes 7 times.

The next set of experiments investigates how the environment scale parameter influences the performance of the IAS-SS strategy. Two environments are considered, Both present the same layout of the environment of Figure 7, but with different sizes. The first is 2 times larger than that

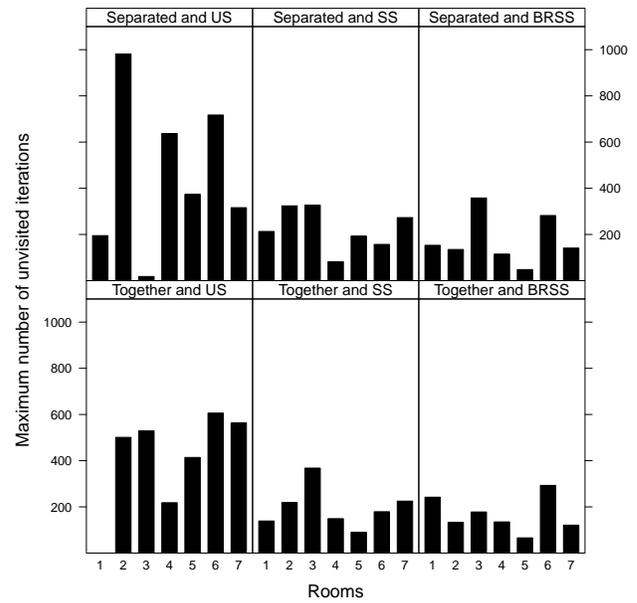


Figure 9. Maximum number of non-visited rooms iterations

environment, while the second is 3 times. The environment used in previous experiments and the two newly defined are called here environments x1, x2 and x3, respectively. The motivation to enlarge the scale of the environment is to assess the suitability of the strategy in sensing all parts of the rooms. Differently from experiments of environment x1, the number of iterations for experiments with environments x2 and x3 are 2000 and 3000, respectively.

The self-adapt trait of the system is visualized through the trajectories of robots of Figure 11. Only the obtained trajectories from simulation of experiments that consider BRSS mechanism and together configuration are shown in order to contrast the slight difference of performed paths. It can be observed that the trajectories are concentrated in a trail when the rooms are small (Figure 11(a)). An explanation for this outcome is the small size of rooms. In this case, the sensor range covers the whole room as the robot enters in the room. While for large regions resultant from environments with duplicated and triplicated scale, the robots move away from the trail to cover the entire environment efficiently (Figures 11(b) and 11(c)). The data presented are from the trial with the median number of SI for each environment configuration.

Analogously to the experiments regarding environment x1, the performances of separated and together configurations for environments x2 and x3 are similar. This is justified by repulsive characteristic of pheromone, which keeps the robots far from each other after while independently of the adopted configuration (separated or together). The Tables V and VI corroborate that there is no strong contrast among

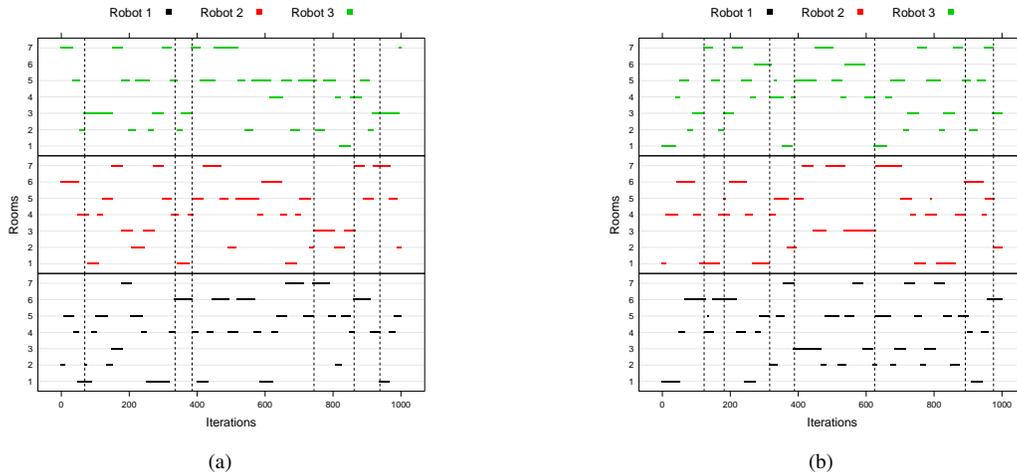


Figure 10. IAS-SS performance according to different configurations for BRSS: (a) separated; (b) together configurations.

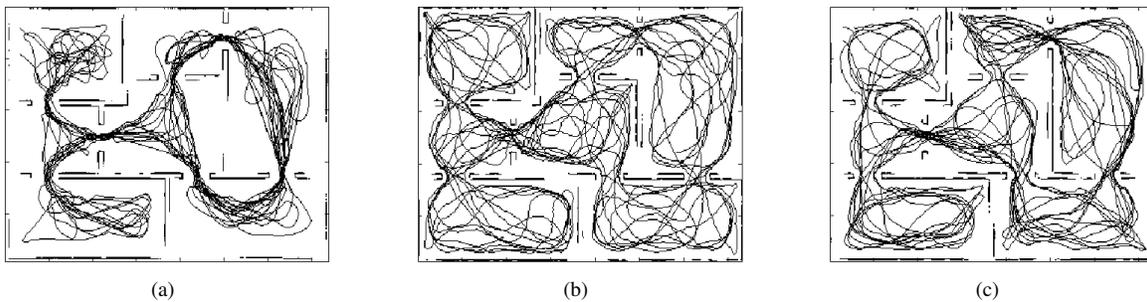


Figure 11. Trajectories of robots during exploration and surveillance tasks for experiments with BRSS mechanism and together configuration: (a) environment x1; (b) environment x2; (c) environment x3.

of performances of separated and together configurations. Regarding the adjustment of steering mechanisms, those configurations with BRSS mechanism yield the best performance when the average of number of SE and average of SI are compared to other mechanisms. The performance of IAS-SS system for the six environment configuration is shown in graphics of Figure 12. The numbers 1, . . . , 6 refer to environment separated configuration with US, SS and BRSS mechanisms; and environment together configuration with US, SS and BRSS mechanisms, respectively.

Maximum periods that rooms are non-visited in experiments of environments x2 and x3 are presented in Figures 13(a) and 13(b), respectively, for the six environment configurations. Data used to plot the graphics are from the trial with the median of average of number of SE and average of SI. It can be noted that regardless which configuration is employed, the system performance improves as long as the mechanism for adjustment of steering direction changes from US to BRSS. The behavior of robots and surveillance intervals for separated and together configurations using BRSS mechanism is clarified in graphics of Figures 14 (environment x2) and 15 (environment x3). For the environment x2, the IAS-SS system with the separated

Table V
PERFORMANCE OF ENVIRONMENT CONFIGURATIONS WITH MECHANISMS FOR ADJUSTMENT OF STEERING DIRECTION FOR ENVIRONMENT X2

Configuration	Uniform Sampling	
	nb. of S.E.	av. of SI.
Separated	2.16 ± 0.75	892 ± 517
Together	1.75 ± 0.95	985.75 ± 576.36
Configuration	Stochastic Sampling	
	nb. of S.E.	av. of SI.
Separated	5.2 ± 1.62	352.72 ± 131.96
Together	4.6 ± 2.01	325.11 ± 94.15
Configuration	Best Ranked Stochastic Sampling	
	nb. of S.E.	av. of SI.
Separated	8.1 ± 2.47	246.8 ± 68.33
Together	8.3 ± 1.88	226.1 ± 59.48

configuration concludes the SE task 9 times and whit the together configuration takes 8 times. While for the environment x3, the SE task is conclude 5 times with the separated configuration and 6 times using the together configuration.

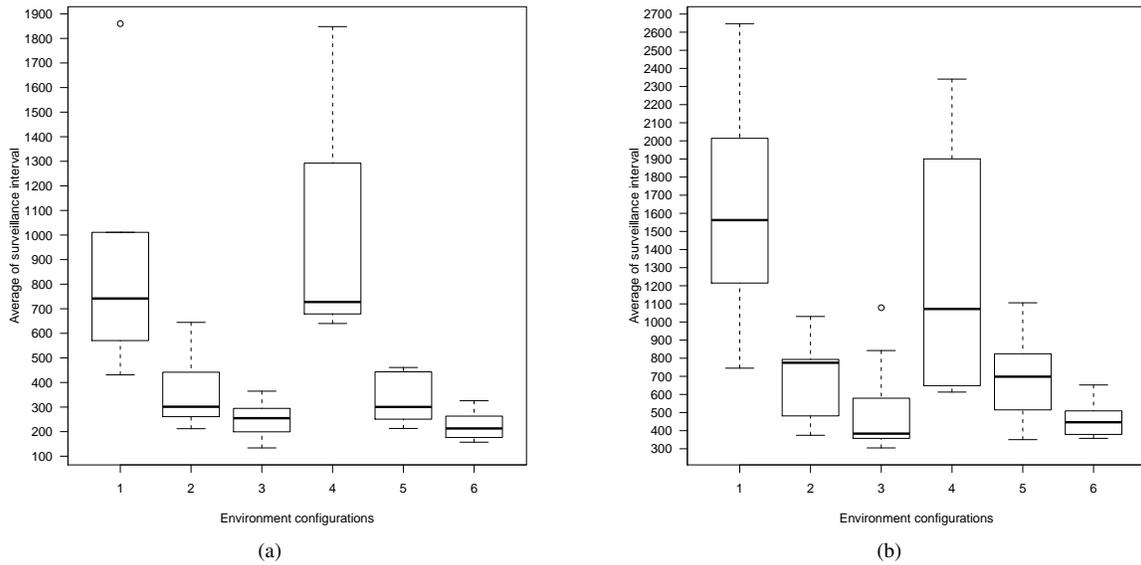


Figure 12. Boxplots of distribution of the average of surveillance intervals for the different adjustment of steering direction mechanisms for: (a) environment x2; (b) environment x3.

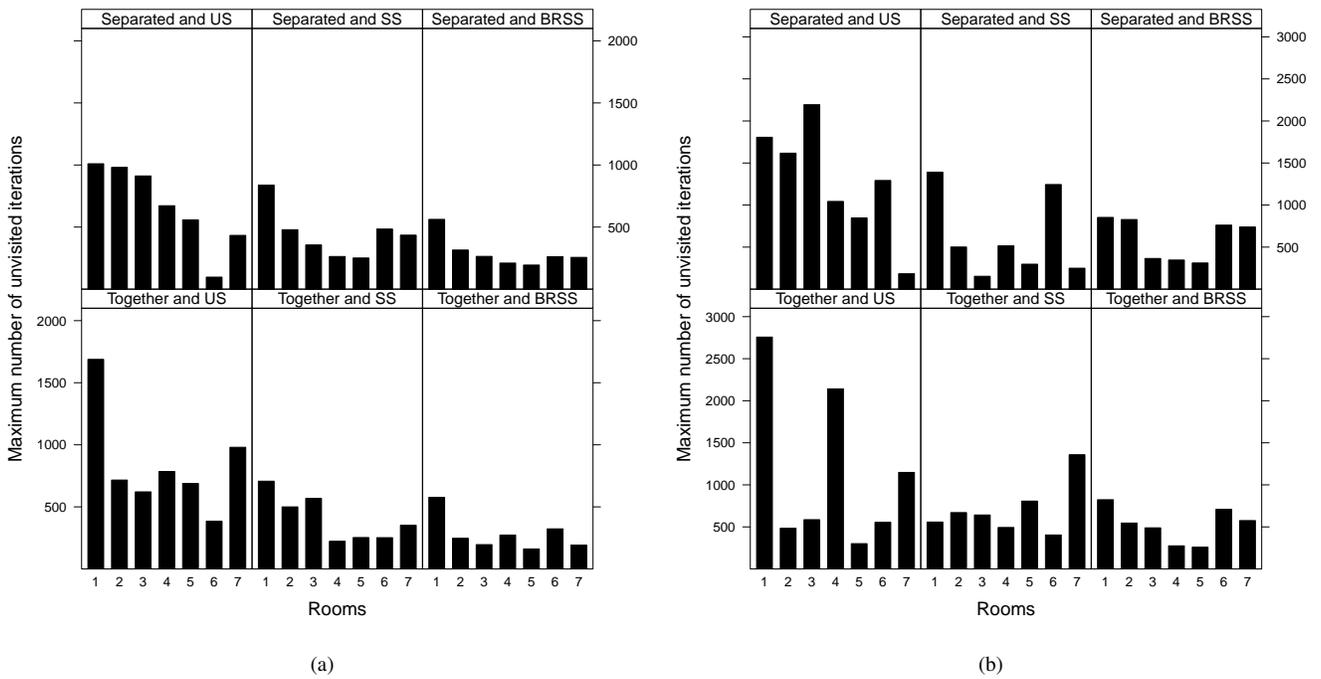


Figure 13. Maximum number of non-visited rooms iterations for: (a) environment x2; (b) environment x3.

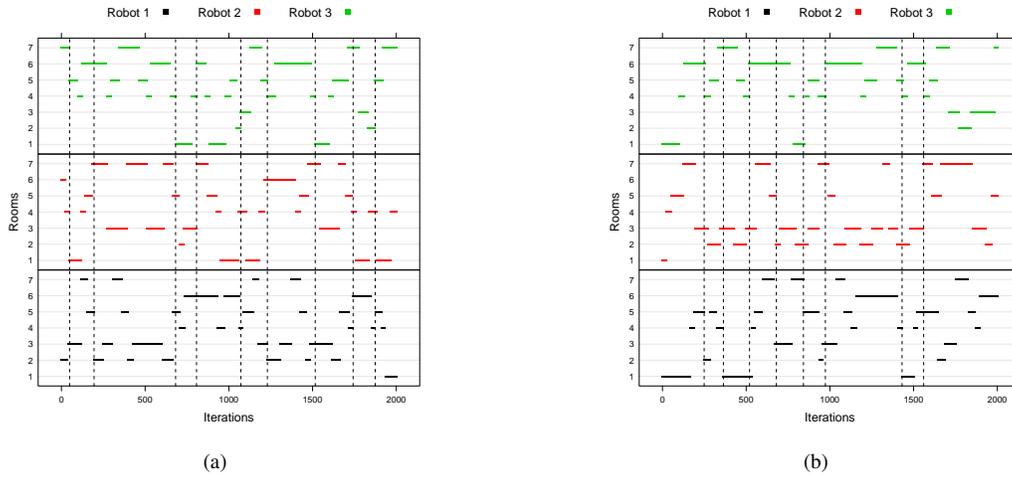


Figure 14. IAS-SS performance according to different configurations for BRSS in environment x2: (a) separated; (b) together configurations.

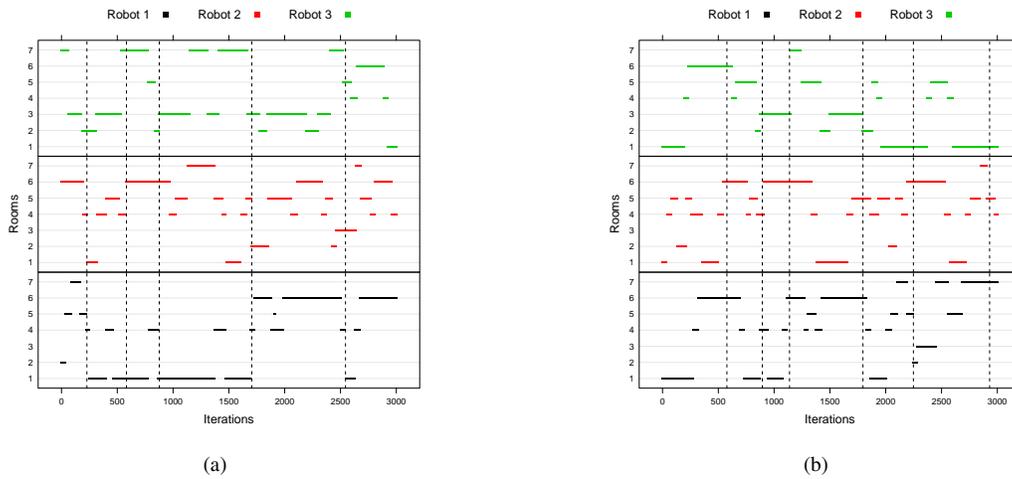


Figure 15. IAS-SS performance according to different configurations for BRSS in environment x3: (a) separated; (b) together configurations.

Table VI
PERFORMANCE OF ENVIRONMENT CONFIGURATIONS WITH MECHANISMS FOR ADJUSTMENT OF STEERING DIRECTION FOR ENVIRONMENT X3

Configuration	Uniform Sampling	
	nb. of S.E.	av. of SI.
Separated	1.37 ± 0.52	1621.87 ± 610.53
Together	1.5 ± 0.57	1274.5 ± 808
Configuration	Stochastic Sampling	
	nb. of S.E.	av. of SI.
Separated	4.33 ± 1.11	675.8 ± 211.17
Together	3.5 ± 1.27	685.28 ± 239.51
Configuration	Best Ranked Stochastic Sampling	
	nb. of S.E.	av. of SI.
Separated	5.44 ± 1.51	519.2 ± 269
Together	6.7 ± 4.47	460.5 ± 100.11

C. Environment Structure

One of characteristic if IAS-SS strategy is the self-adapt. It is emphasized in this section. Following experiments aim

at analyzing the performance of exploration and surveillance tasks independently of environment structure. To investigate this characteristic, distinct environment structures are designed from a rectangular space divided virtually in 10 rooms as illustrated in Figure 16(a). The connectivity among adjacent rooms is represented by a graph (Figure 16(b)). Accesses that connect rooms are partially or totally blocked by obstacles, generating different environments. According to this process, ten models of environment are considered, such that, each environment is associated to a complexity level.

Complexity level is measured according to number of options to travel the environment (among rooms), that is, through graph structure resultant from connection among rooms. The more path options to reach a specific region are available, the complexity level of environment is higher. For environments of Figures 16(c) and 16(d), the graph structure is the same of the graph of Figure 16(b), hence, the com-

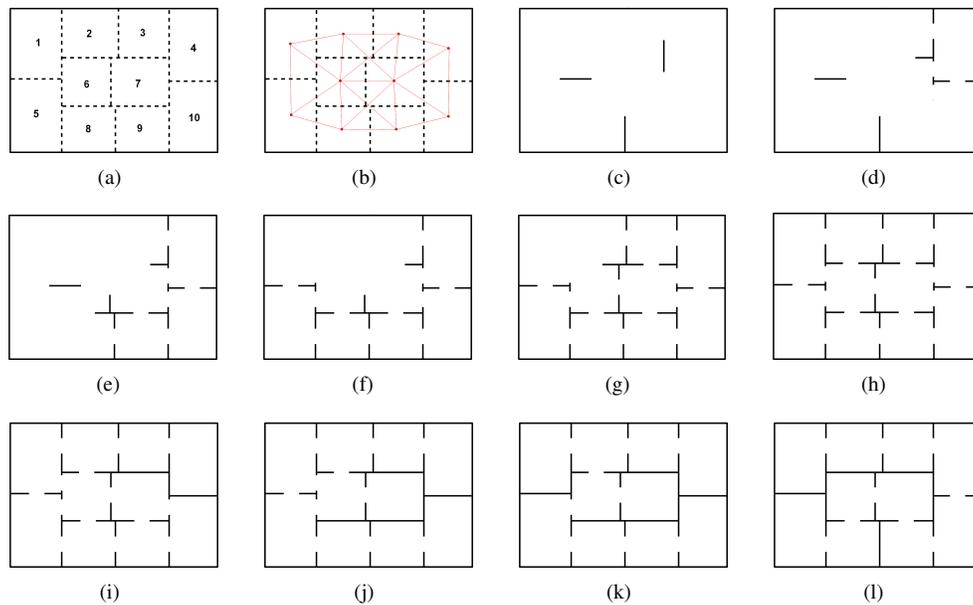


Figure 16. Environment models: (a) environment divided in rooms; (b) connection graph among rooms; (c)-(l) environment from #1 to #10.

plexity is low. As obstacles are inserted into environments blocking the passage among rooms, the respective edges of graph are removed and, thus, the complexity is higher.

Since there are ten rooms, four robots are considered for experiments to assign at least two rooms to each robot. This forces the robots travels long distances increasing the likelihood find challenging situations as obstacles. All robots start at room 1.

Although it is clear that the exploration time decreases as complexity level increases, the surveillance task is accomplished even with a restricted number of path options. This emphasizes that environment structure is not a factor that impedes the tasks to execute. Even robots in environments with higher complexity level can carry out the tasks. The environment sensing (SE) is completed independently of the environment structure. As general behavior of the system, the length of SI period is increased while the complexity level of environment increases. Also, as consequence of the higher complexity, the number of completed SE is smaller. This can be observed in Table VII. The average of number of SE increases and the average of SI presents a strong decreasing tendency, which is not monotonic due to the random nature of experiments. Therefore, it is observed that the system self-adapt according to changes in the environment model. A more detailed view of results of table, regarding the average of SI, is presented in the Figure 17. It shows the boxplots of the distribution of the performance.

The self-adapt trait of the system is visualized through the trajectories of robots and maps of average of amount of pheromone of Figure 18 for some environment models in order to contrast the high difference of complexity level among them. It can be observed that the trajectories are concen-

Table VII
PERFORMANCE OF CONFIGURATION WITH BRSS MECHANISM AND INCREASING THE COMPLEXITY LEVEL

Environment	Number of SE	Average of SI
#1	17 ± 3	57.46 ± 10.41
#2	15.66 ± 2.08	61.9 ± 7.83
#3	13.66 ± 0.57	70.76 ± 6.27
#4	15 ± 2	63.77 ± 8.91
#5	12.66 ± 1.52	77.29 ± 11.58
#6	11 ± 1.73	87.65 ± 19.59
#7	9 ± 0.01	97.75 ± 7.7
#8	7.66 ± 57.73	114.17 ± 23.82
#9	7.66 ± 57.73	119.49 ± 3.88
#10	7.33 ± 1.52	115.23 ± 24.9

trated in a trail when the rooms are small. An explanation for this outcome is the small size of rooms. In this case, the sensor range covers whole the room. While for large regions resultant from junction rooms in environments #1 and #3, the robots move away from the trail to cover the entire environment efficiently. The data presented are from the trial with the median number of SI for each environment.

D. Number of robots

This section discusses about the relation between the size of environment and number of robots. Indeed, higher number of robots is, more regions are explored and monitored simultaneously, so that, few or no regions are empty for long period. Since robots behavior is based on inverse of ant algorithm, the probability of one robot explorer and monitor large environments is higher. However, it may take a long time. In order to evaluate the performance of motion coordination and the efficiency of surveillance task, experiments are carried out with an increasing number of

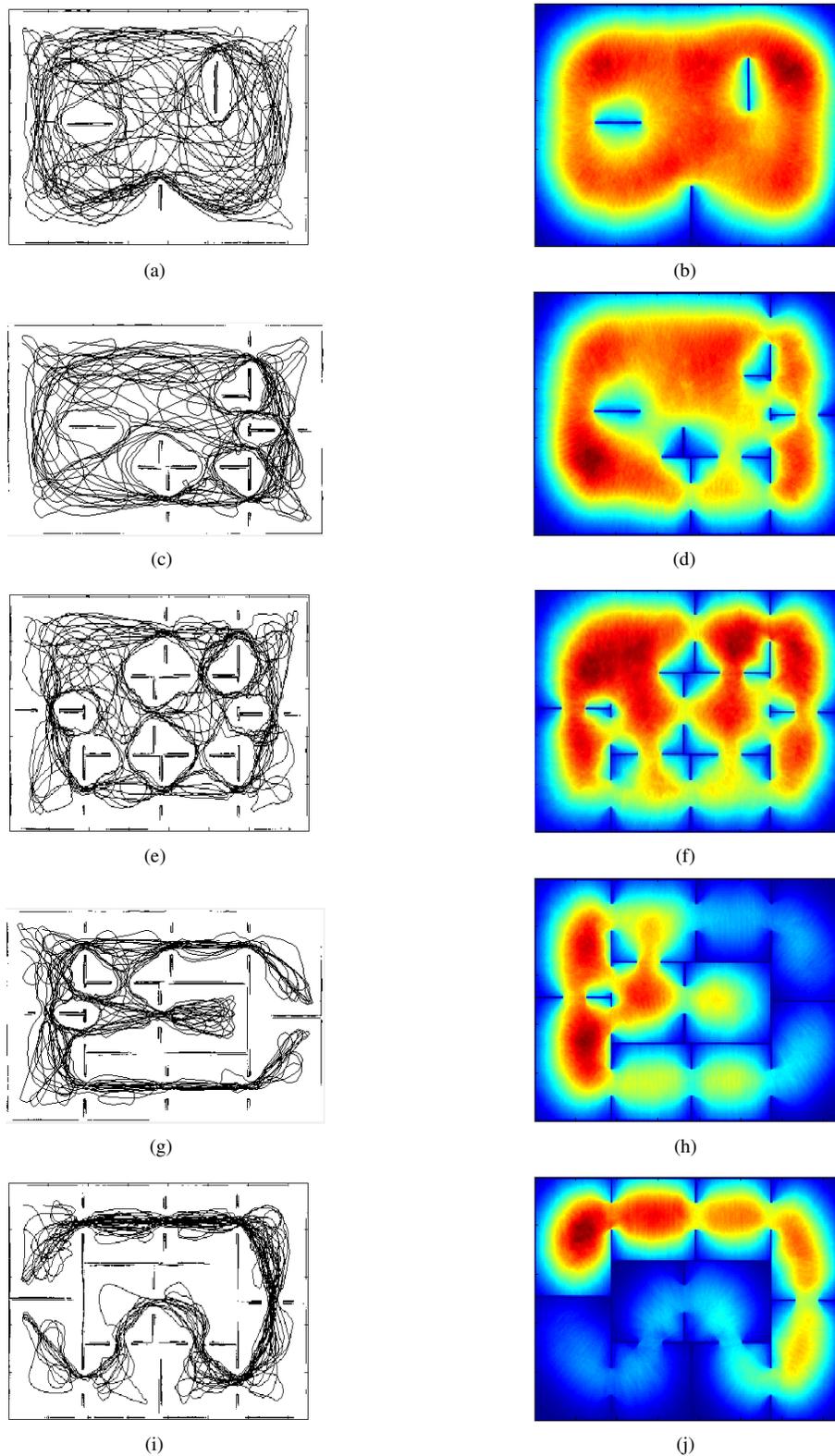


Figure 18. Trajectories of robots and maps of average of amount of pheromone according to distinct environment structures: (a)-(b) #1; (c)-(d) #3; (e)-(f) #5; (g)-(h) #8; (i)-(j) #10.

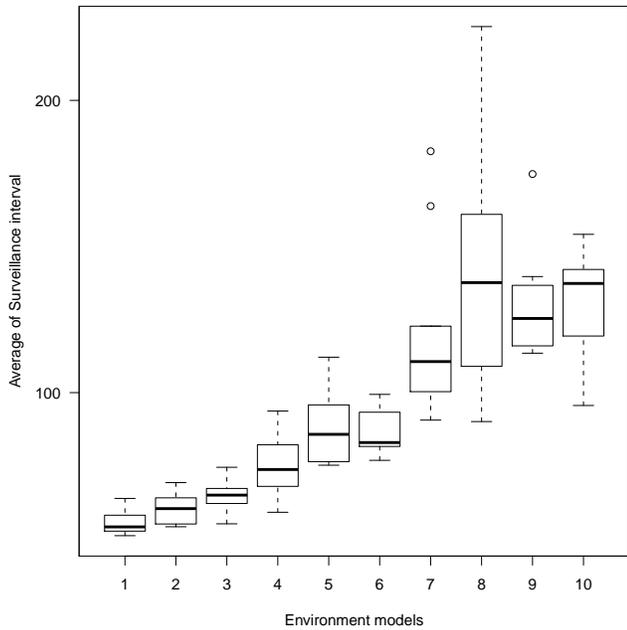


Figure 17. Boxplots of distribution of the average of surveillance intervals for different degree of complexity of environment.

robots in environment of Figure 19. Since BRSS mechanism presented better performance than US and RS mechanisms in previous experiments, this mechanism is adopted to analyze the efficiency of the exploration and surveillance tasks while the number the robots increases. All added robots are placed at room 1.

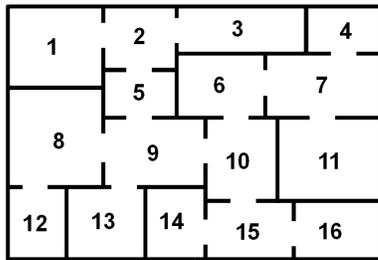


Figure 19. Environment structure

Although it is clear that the time to explore decreases as number of robots increases, the surveillance task is accomplished even with a restricted number. This emphasizes that number of robots is not a factor to limit the size of the explored environment. Even few robots are able to monitor large areas. The environment sensing (SE) is completed independently of the number of robots. However, as general behavior of system, the length of SI period is reduced while the number of robots increases. Also, as consequence of the addition of robots, there are more completed SE. This can be observed in Table VIII. As the number of robots increases, the average of number of SE increases together.

Conversely, the average of SI presents a strong decreasing tendency, which is not monotonic due to the aleatory nature of experiments. Therefore, it is observed that the system self-adapt according to the number of robots. A more detailed view of results of table, regarding the average of SI, is presented in the Figure 20. It shows the boxplots of the distribution of the performance for the 10 trials.

Table VIII
PERFORMANCE OF TOGETHER CONFIGURATION WITH BRSS MECHANISM FOR INCREASING NUMBER OF ROBOTS

Number of robots	Number of SE	Average of SI
2	0.5 ± 0.7	598.25 ± 196.62
3	1.1 ± 0.57	621.44 ± 179.88
4	1.5 ± 0.97	412.56 ± 190.89
5	1.5 ± 0.97	504.77 ± 239.93
6	2.4 ± 0.51	220.60 ± 65.26
7	3.2 ± 0.78	174.01 ± 58.25
8	3.4 ± 0.98	155.97 ± 59.30
9	3.9 ± 1.28	137.94 ± 61.57
10	5.3 ± 1.5	118.75 ± 20.92
11	5.7 ± 2	115.98 ± 44.14
12	6.4 ± 1.89	107 ± 38
13	6.9 ± 2.42	108.4 ± 46.47
14	8.2 ± 1.68	88.04 ± 21.93
15	8.3 ± 2.78	92.02 ± 40.23

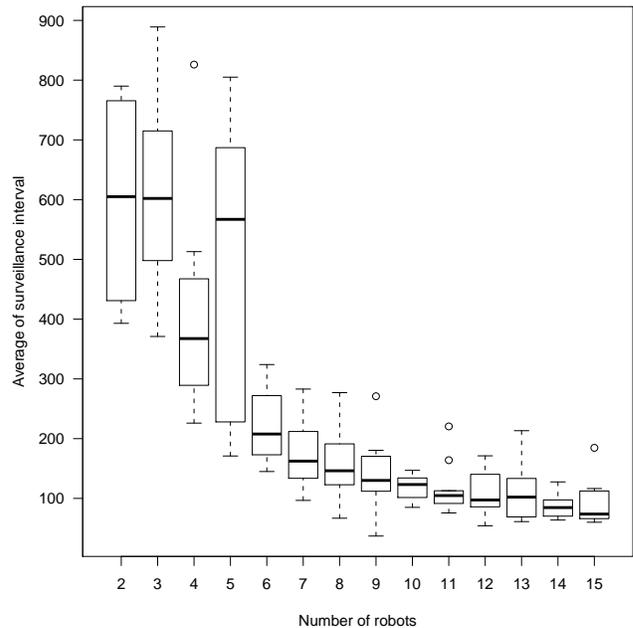


Figure 20. Boxplots of distribution of the average of surveillance intervals for the different mechanisms for adjustment of steering directions

The experiments performed are to show that the autonomous proposal gets to execute the surveillance task satisfying the constraints set up in the equation 2. However, the minimization of the objective function is not considered here. In this step, the work is only to verify the surveillance capability.

E. For Real Robots

For the validation of IAS-SS strategy in a real robots platform, two approaches are suggested. The first one is to equip physical robots with devices for releasing some chemical and odour sensors. As long as the robots navigate, they left this substance in the frontal regions to mark them as explored regions. Through odour sensors, the robots are able to detect the regions more attractive, i.e., the regions with low amount of the substance. The second one only considers distance sensors, disregarding the presence of the odour sensors. Then, applying the mapping method, occupancy-grid, as mentioned in Section V, the generated cells could be used for hosting a value that indicates the amount of deposited pheromone. Hence, the amount of pheromone would exist in a virtual way. The virtual pheromone releasing and detecting areas correspond to the range area of the distance sensors. However, in the last approach, a localization method [34] would be need in order to each robot to built its own local map and, accordingly, to join it with the local maps of another robots. Thus, using less sensors, the strategy is able to explore the entire environment and to perform the surveillance task. The implementation in real robots makes part of future works.

VI. CONCLUSIONS AND FUTURE WORKS

In this work, it was proposed a new bio-inspired distributed coordination strategy, named IAS-SS, for multiple robot systems applied for exploration and surveillance tasks. The strategy is based on swarm theory, specifically the ant system theory. The repulsive character as a function of the deposited pheromone quantity determines the dynamic of behavior of the agents (robots) and stimulates them to be spread out by the environment. As a consequence, the agents get to monitor the entire environment continuously, visiting regions not recently visited. Furthermore, other contributions can be highlighted, such as, the development of a decentralized strategy, where the robots are independent agents that define their steering direction without an external influence; a reactive strategy, in which the only information necessary for the robots is extracted from amount of pheromone and, finally, a pheromone trail is not generated, since the deposit of pheromone occurs only in areas covered by distance sensors (i.e., frontal areas to the robot).

Although the strategy is very simple compared to other environment exploration strategies, both exploration and surveillance tasks were efficiently performed. A set of experiments were done for analysing the performance of the proposed system. Experiments considered two performance criteria: the average of the numbers of surveillance epochs and average of the surveillance time intervals. Four parameters, namely: start position, number of robots, environment scale and environment structure; stress the strategy capabilities. Two versions of the IAS-SS strategy were considered and compared with a totally random strategy.

The IAS-SS strategies presented significantly a superior performance. Some characteristics of these strategies were noted, such as, they are not dependent on the knowledge of the environment structure and they are robust in regard to the number of robots. These strategies kept robots well separated guiding them toward regions not recently visited. The advantage of the bioinspired strategy proposed resides in, among other aspects: simple conceptual ideas, reduced computation complexity, real time operation and efficiency.

It is important to say that calculate the complexity of proposed algorithm is a tedious task. Since the performance criterium is the number of SE and average of SI, the obtained results present distinct performance due to changes of structure of environments, number of robots and initial position configuration. Therefore, there are many combinations to establish the coordination strategy. In addition, the approaches about monitoring found in literature emphasize graphs to define the routes of robots. Therefore, there is no approach similar to the present proposal in order to summarize a suitable comparison of complexity.

As future works experiments will be designed to investigate two aspects: how different pheromone releasing mechanisms influence the performance of the IAS-SS system; and to investigate the adaptation capability when some robots fail. Moreover, a method to join maps will be conceived and integrated to IAS-SS system in order to apply it in real robots. Thus, it is need to develop a communication device to support change information about mapping. This device, coupled to robots, will able to identify other robots and transfer data through wireless network. In addition, more complex surveillance tasks, e.g., those that a strange agent invades the environment, will be investigated. In this case, a vision system with tracking ability is essential.

ACKNOWLEDGMENT

The authors would like to thank Sao Paulo Research Foundation (FAPESP) and National Council for Scientific and Technological Development (CNPq) for their support.

REFERENCES

- [1] R. Calvo, J. R. de Oliveira, M. Figueiredo, and R. A. F. Romero. "Inverse ACO Applied for Unknown Environment Exploration". In: The Third International Conference on Advanced Cognitive Technologies and Applications, 2011, Rome, Italy. Proceedings of COGNITIVE'2011, p. 142-147, 2011.
- [2] J. G. Bellingham, and M. Godin, "Robotics in Remote and Hostile Environments". *Science*, vol. 318, pp. 1098-1102, 2007.
- [3] H. H. Schmitt, "From the Moon to Mars". *Nature*, vol. 301, pp. 36-43, 2009.
- [4] F. Mazzini, D. Kettler, J. Guerrero, and S. Dubowsky. "Tactile Robotic Mapping of Unknown Surfaces, With Application to Oil Wells". *IEEE Transactions on Instrumentation and Measurement*, vol. 60, pp. 420-429, 2011.

- [5] X. L. Long, J. P. Jiang, and K. Xiang. "Towards Multirobot Communication". IEEE International Conference on Robotics and Biomimetics, 2004. ROBIO 2004, pp. 307-312, 2004.
- [6] A. Speranzon. "Coordination, Consensus and Communication in Multi-Robot Control Systems". PhD Thesis, Royal Institute of Technology, Stockholm/Sweden, 2006.
- [7] C. Liu, Y. Ma, and C. Liu. "Cooperative Multi-robot Map-Building Under Unknown Environment". Proceedings of the 2009 International Conference on Artificial Intelligence and Computational Intelligence, vol. 3, pp. 392-396, 2009.
- [8] L. Andersson, and J. Nygård. "On Multi-robot Map Fusion by Inter-robot Observations". In proceedings of 12th International Conference on Information Fusion, 2009.
- [9] B. Yamauchi. "A frontier-based approach for autonomous exploration". Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation, pp. 146-151, 1997.
- [10] F. Pasqualetti, A. Franchi, and F. Bullo. "On optimal cooperative patrolling". IEEE Conf. on Decision and Control, Atlanta, GA, USA, pages 7153-7158, December 2010.
- [11] D. Scheidt, and J. Stipes. "Cooperating unmanned vehicles". Proceedings of the 2005 IEEE Networking, Sensing and Control, pp. 326-331, 2005.
- [12] J. Stipes, R. Hawthorne, D. Scheidt, and D. Pacifico. "Cooperative Localization and Mapping". Proceeding of the IEEE, Networking, Sensing and Control, pp. 596-601, 2006.
- [13] A. Marjovi, J. G. Nunes, L. Marques, and A. de Almeida. "Multi-robot exploration and fire searching". Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems, pp. 1929-1934, 2009.
- [14] A. Howard, M. J. Mataric, and G. S. Sukhatme. "Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem". Proceeding of the 6th International Symposium on Distributed Autonomous Robotics Systems, pp. 299-308, 2002.
- [15] J. Tan, N. Xi, W. Sheng, and J. Xiaov. "Modeling multiple robot systems for area coverage and cooperation". Proceedings of the 2004 IEEE International Conference on Robotics and Automation, pp. 2568-2573, 2004.
- [16] Q. Jiang. "An improved algorithm for coordination control of multi-agent system based on r-limited voronoi partitions". Automation Science and Engineering, 2006. CASE '06. IEEE International Conference on, pp. 667-671, 2006.
- [17] F. WeiXing, W. KeJun, Y. XiuFen, and G. ShuXiang. "Novel Algorithms for Coordination of Underwater Swarm Robotics", *Proc. of IEEE Int. Conf. on Mechatronics and Automation*, pp. 654-659, 2006.
- [18] R. A. Freeman, P. Yang, and K. M. Lynch. "Distributed estimation and control of swarm formation statistics", *American Control Conf.*, 7 pp. 749-755, 2006.
- [19] F. Kobayashi, N. Tomita, and F. Kojima. "Re-formation of mobile robots using genetic algorithm and reinforcement learning", *SICE 2003 Annual Conf.*, vol. 3, pp. 2902-2907, 2003.
- [20] L. Barnes, W. Alvis, M. A. Fields, K. Valavanis, and W. Moreno. "Swarm Formation Control with Potential Fields Formed by Bivariate Normal Functions", *14th Mediterranean Conf. on Control and Automation*, pp. 1-7, 2006.
- [21] J. M. Hereford. "A Distributed Particle Swarm Optimization Algorithm for Swarm Robotic Applications", *Proc. of IEEE Congress on Evolutionary Computation*, pp. 1678-1685, 2006.
- [22] M. Hess, M. Saska, and K. Schilling. "Formation driving using particle swarm optimization and reactive obstacle avoidance", *Proc. of 1st IFAC Workshop on Multivehicle Systems (MVS)*, pp. 32-37, Lisboa, Portugal, 2006.
- [23] M. Dorigo, M. Birattari, and T. Stützle. "Ant Colony Optimization Artificial Ants as a Computational Intelligence Technique". IEEE Comput. Intell. Mag, pp. 28-39, 2006.
- [24] R. Calvo, J. R. de Oliveira, M. Figueiredo, and R. A. F. Romero. "A Distributed, Bio-Inspired Coordination Strategy for Multiple Agent Systems Applied to Surveillance Tasks in Unknown Environments". In: International Joint Conference on Neural Networks, 2011, San Jose, CA, USA. Proceedings of IJCNN'2011. Los Alamos : IEEE Press, p. 3248-3255, 2011.
- [25] R. Calvo, J. R. de Oliveira, M. Figueiredo, and R. A. F. Romero. "Bio-inspired coordination of multiple robots systems and stigmergy mechanisms to cooperative exploration and surveillance tasks". In: IEEE 5th International Conference on Cybernetics and Intelligent Systems and on Robotics, Automation and Mechatronics, 2011, Qingdao, China. Proceedings of CIS-RAM'2011, p. 223-228, 2011.
- [26] M. Dorigo. "Optimization, learning and natural algorithms". PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, 1992.
- [27] E. Bonabeau, M. Dorigo, and G. Theraulaz. "Inspiration of optimization from social insect behavior". *Nature*, pp. 39-42, 2000.
- [28] M. Dorigo, G. Di Caro, and L. Gambardella. "Ant Algorithms for Discrete Optimization". Technical Report IRIDIA/98-10, Universite Libre de Bruxelles, Belgium, 1999.
- [29] D. Yingying, H. Yan, and Jiang Jingping. "Multi-robot cooperation method based on the ant algorithm". Proceedings of the 2003 IEEE Swarm Intelligence Symposium, pp. 14-18, 2003.
- [30] B. Christian. Review of "Ant colony optimization" by M. Dorigo, and T. Stützle, MIT Press, Cambridge, MA, 2004. *Artif. Intell.*, pp. 261-264, 2005.
- [31] L. A. Zadeh, "Fuzzy Sets", *Information and Control*, Vol. 8, 338-353, 1965.
- [32] R. Calvo, M. Figueiredo, and R. A. F. Romero. Autonomous cognition and reinforcement learning. In: IEEE International Joint Conference on Neural Network, Barcelona, Spain. Proceedings of IJCNN'2010, p. 1-8, 2010.

- [33] A. Elfes. "Using Occupancy Grids for Mobile Robot Perception and Navigation". *Computer*, issn 0018-9162, vol 22, pp. 46-57, Los Alamitos, CA, USA, 1989.
- [34] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. "Robust Monte Carlo localization for mobile robots". *Artificial Intelligence*. vol. 128, n. 1-2, pp. 99 - 141 2000.

Quality Attributes for Web Services: A Model-based Approach for Policy Creation

Alexander Wahl, Bernhard Hollunder, and Varun Sud

Department of Computer Science

Furtwangen University of Applied Science

Furtwangen, Germany

alexander.wahl@hs-furtwangen.de, bernhard.hollunder@hs-furtwangen.de, varun.sud@hs-furtwangen.de

Ahmed Al-Moayed

BI/ HANA Department

Adweko GmbH

Walldorf, Germany

ahmed.al-moayed@adweko.com

Abstract—Service-oriented architectures (SOA) define a conceptual framework for the creation and integration of enterprise applications. Within an SOA, the core functionality is realized by distributed services, which are typically composed to support the required business processes. Today, Web services are the predominant technology to implement and deploy services in heterogeneous environments. In many business domains, Web services must exhibit quality of service (QoS) attributes such as security, performance, scalability, and accounting. Currently, there is only limited support for the assignment of QoS attributes to Web services, though. In this paper, we present a model-based approach for deriving policies from a QoS model. Our solution covers the modeling of QoS attributes based on a meta-model for quality attributes, the generation of a graphical user interface to configure the modeled QoS attributes, and the transformation into policy descriptions. Finally, these policies will be assigned to the target Web services. To highly automate our approach, we apply techniques from model-driven development such as model-to-model and model-to-code transformations. As a consequence, our solution reduces the cost and effort when creating QoS-aware Web services.

Keywords-Service-oriented architecture; Web services; QoS meta-model; model-to-model transformation; model-to-code transformation; WS-Policy.

I. INTRODUCTION

Service-oriented architectures (SOA) refer to a system architecture that provides applications and software components as reusable and interoperable services with well-defined business functionalities. In most deployment settings, the services must also address non-functional requirements such as security, performance, and accounting in order to guarantee predefined quality of service (QoS) attributes the overall business applications must fulfill. As an enterprise typically refers to a variety of internal and

This is a revisited and substantially augmented version of “An Approach to Model, Configure and Apply QoS Attributes to Web Services”, which appeared in the Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA 2011) [1].

external service providers, it is crucial to explicitly assign QoS attributes to the underlying Web services.

In the literature, several policy languages have been proposed to formally specify QoS attributes for particular technical or business domains. With the WS-Policy specification [2], there exists a well-known and widely used framework for defining QoS attributes for Web services. Basic building blocks in WS-Policy are so-called assertions, where a single assertion may represent a domain-specific capability, constraint or requirement. In order to create valid WS-Policy descriptions for non-trivial scenarios, technical knowledge regarding the design of WS-Policy assertions and the underlying policy grammar is required (see, e.g., [3], [4]).

However, a developer may not necessarily acquire this knowledge, but should be enabled to easily assign QoS attributes to the Web service under development.

In this paper, we present a model-based approach to specify, configure, and assign QoS attributes to Web services. Given a QoS meta-model, the “QoS profile developer” creates instances of the meta-model, which formalize QoS attributes for dedicated domains such as security or performance. Such models, also called QoS profiles, are reusable assets and can be applied to different Web services by the “Web service developer”. By means of model transformations, a graphical user interface (GUI) is generated, which is used by the developer to adjust the predefined QoS attributes for the specific deployment context of the Web service. Eventually, the refined QoS profiles are automatically translated into corresponding descriptions of well-known policy languages such as WS-Policy.

It should be noted that Integrated Development Environments (IDE) such as Eclipse, NetBeans, and Visual Studio provide specific project types for the construction of Web services. These environments automate many activities such as code compiling, WSDL interface generation, creation of proxy objects and code deployment. However, currently

there is only limited support for developing QoS-aware Web services, i.e., Web services with well-defined QoS attributes. In particular, these environments are either hard to extend or are restricted to certain policy domains such as WS-SecurityPolicy [5] and WS-ReliableMessaging [6]. To our best knowledge, there is no framework covering the following features:

- A simple, but powerful QoS meta-model for formalizing arbitrary QoS attributes.
- An easy way to create QoS profiles (i.e., collections of QoS attributes) for Web services.
- The automatic creation of a graphical user interface, which allows the developer to configure the modeled QoS attributes for each designated Web service.
- Automatic transformation of the configured QoS profile into equivalent policy descriptions.
- Assignment of the created policy description to the Web service under development.

In this work, we elaborate the conceptual and technical foundations of such a framework. We also describe our proof of concept implementation, which demonstrates the feasibility of the approach. Our solution is a further step to reduce the developing effort and the costs of creating QoS-aware Web services.

This paper is structured as follows. The next section describes the problem addressed in this paper in more detail and sketches our solution strategy. After introducing the solution architecture in Section III, the successive sections focus on particular elements: Section IV introduces the QoS meta-model followed by the QoS profile (Section V) and the graphical user interface for configuring QoS attributes (Section VI and VII). In Section VIII, the generation of policy descriptions is elaborated. A description of the proof of concept implementation is given in Section IX. Then, a discussion on related work (Section X) and future work (Section XI) is given, followed by a conclusion.

II. PROBLEM DESCRIPTION AND SOLUTION STRATEGY

Developing QoS-aware Web services is a strenuous task for Web service developers. Typically, QoS attributes are hardcoded into the Web service business logic increasing code complexity. Implementing QoS attributes in the source code also decrease the degree of reusability of Web service for different deployment settings and flexibility to react on changing QoS requirements. Web service developers explicitly require knowledge of policy languages, e.g., WS-Policy, to create and apply policies to Web services. They also require knowledge of associated policy grammars such as policy-domain specific tags, elements, nesting rules, rules of operations and operators and policy expression creation rules. Such knowledge is not always available to the Web service developers. Currently available solutions and support to create and apply QoS attributes to the Web services are

either limited to certain domains or specific to development environments.

In this paper, we present an approach to answer the following questions and challenges that emerge with such QoS-aware Web services:

- Is it possible to model, configure and apply not only standardized QoS attributes but also project specific QoS attributes to Web services in an easy, extendable and flexible manner?
- Is there a solution to enable quick development of QoS-aware Web services irrespective of the Web service implementation language, business logic and policy domains?
- Moreover, is there a solution to handle and enable frequent changes in business requirements with respect to non-functional requirements?
- Can the time, complexity and effort in designing, modeling, creating and applying QoS attributes to the Web services be reduced?

This paper offers a solution, a tool chain, which automates and simplifies modeling, configuring and applying of QoS attributes to Web services. Figure 1 describes the working of our proposed solution with the actors, components and processes involved.

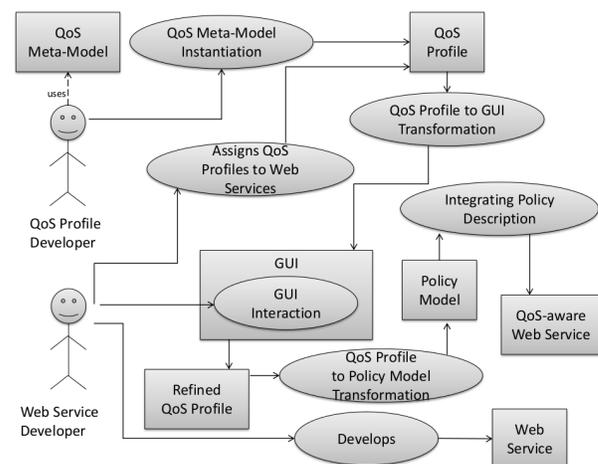


Figure 1. Use case diagram describing our solution strategy.

As shown in the figure, there are two distinctive roles: the QoS profile developer and Web service developer performing their concerned tasks. QoS profile developer uses the QoS meta-model to create a so-called QoS profile, which forms a reference between the Web services and the QoS attributes. Examples of such QoS profiles are security profile and performance profile for a Web service.

By undergoing certain set of transformations, a GUI is generated from a QoS profile. Web service developer now interacts with the generated GUI to create a refined QoS profile based on certain business requirements. The Web

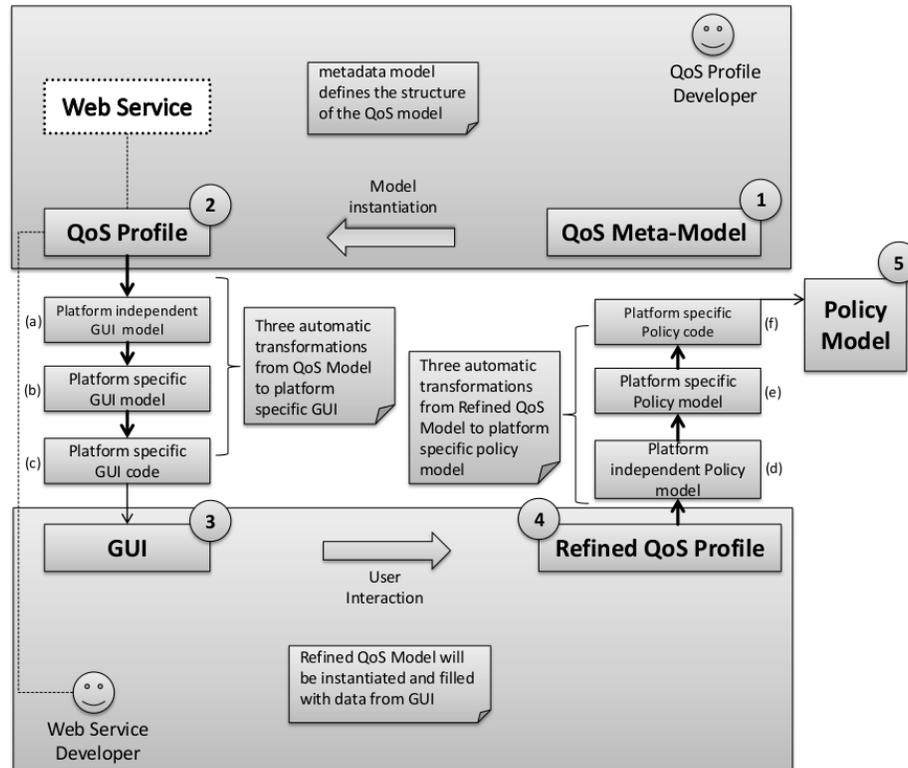


Figure 2. An approach to model, configure and apply QoS attributes to Web services.

service developer later assigns the refined QoS profile to the concerned Web service.

Refined QoS profile follows a set of transformations to create a policy model. Policy model is the starting point to produce policy descriptions to be integrated into the Web services to finally generate QoS-aware Web services. By introducing this solution strategy, we can answer the questions described in the problem description in the following manner:

- QoS meta-model component allows modeling, configuring and applying QoS attributes of not only standardized but also project specific QoS attributes in an easy extendable and flexible manner.
- QoS meta-model, QoS profile, refined QoS profile and policy model are components of the solution strategy. QoS profile, refined QoS profile and policy model undergo transformations to generate business specific solutions.
- The mentioned transformations also allow re-processing of new technical and business specifications at different components of the solution strategy.
- Separating the tasks of QoS profile developer and Web service developer reduces the effort, time and complexity of designing, modeling, creating and applying QoS attributes to Web services.

III. APPROACH

Figure 2 shows the solution architecture in more details describing our approach. The first component is the QoS meta-model (see (1) in Figure 2). It describes exactly how the QoS profiles are created. It is simple, extensible, easy to understand and expressive enough to model arbitrary QoS attributes.

The second component is the QoS profile (see (2) in Figure 2), which is an instance of QoS meta-model. It offers the QoS profile developer a way to model QoS attributes within QoS categories that are already defined in the QoS meta-model. As we will see in Section VII, with this meta-model, we will be able to model different QoS attributes including QoS attributes such as reliable messaging and performance.

Once a QoS profile has been instantiated, the third component of the solution, a GUI is generated (see (3) in Figure 2) based on certain transformation rules. The transformations process essential information from the QoS profile and depending on the elements in the QoS profile generate the GUI. The main purpose of the GUI is to provide Web service developer an interface to refine and configure available QoS profiles. For example, a Web service developer can choose and specify specific encryption algorithm for a QoS security profile.

After a successful user interaction with the GUI, the fourth component of the solution, i.e., a refined QoS profile (see (4) in Figure 2) is obtained. The refined QoS profile is the actualized version of the QoS profile, which is again transformed into the policy representation, i.e., the fifth component of the solution (see (5) in Figure 2). The policy model is the final step towards applying QoS attributes as policies to Web services.

In general, there are six transformations taking place from QoS profile to policy model generation:

- An automated transformation from QoS profile to platform independent GUI model (GUI PIM). This enables easy extension of the solution to different GUI technologies such as Swing, SWT and WPF.
- An automated transformation from GUI PIM to platform specific GUI model (GUI PSM). GUI PSM is specific to modelled QoS attributes.
- An automated transformation from GUI PSM to platform specific GUI code.
- An automated transformation from refined QoS profile to platform independent policy model (Policy PIM). This enables easy extension of the solution to different policy formalisms such as WS-Policy [2] and XACML [7].
- An automated transformation from Policy PIM to platform specific policy model (Policy PSM). Policy PSM is specific to a policy domain selection of which is based on project or business requirements.
- An automated transformation from platform specific Policy model to platform specific policy description. The transformation reduces complexity, time and effort to generate policy descriptions.

In the next sections, we will describe the components shown in Figure 2 in more details.

IV. QoS META-MODEL

There are several QoS meta-model proposals which can be used to define and apply QoS profiles for Web services. Malfatti [8] introduced a suitable meta-model for our approach. Figure 3 shows the QoS meta-model used in our solution which is a slightly modified version of Malfatti. It is extensible and expressive enough to model standardized and arbitrary QoS attributes. The meta-model is created in Eclipse Modeling Framework (EMF-core), a powerful tool for designing models and their runtime support [9]. QoS profile developer uses this QoS meta-model to instantiate QoS profiles with QoS attributes. The basic elements of the QoS meta-model are:

- Service:** Name of the Web service to apply policies. The Web service can have zero or more `QoSCategory` elements.
- QoSCategory:** Defines categories with which quality criterions are grouped. Examples of a `QoSCategory` could be security, reliability and performance. Each `QoSCategory` has one or more `QoSParameters`.

- QoSParameter:** It describes the quality criterions, e.g., `Inactivitytimeout` is a `QoSParameter` for reliability category. Each `QoSParameter` has exactly one `QoSAgreedValue` and a `QoSMetric` associated with it.
- QoSAgreedValue:** The value of the criterion is defined in this element, e.g., the value 20 for `Inactivitytimeout`. This element can also be extended with `QoSProperty` elements.
- QoSMetric:** This element specifies a unit with which the value of `QoSAgreedValue` element is measured e.g., “seconds” for the `QoSAgreedValue` 20 which is associated with the `QoSParameter` `Inactivitytimeout`.

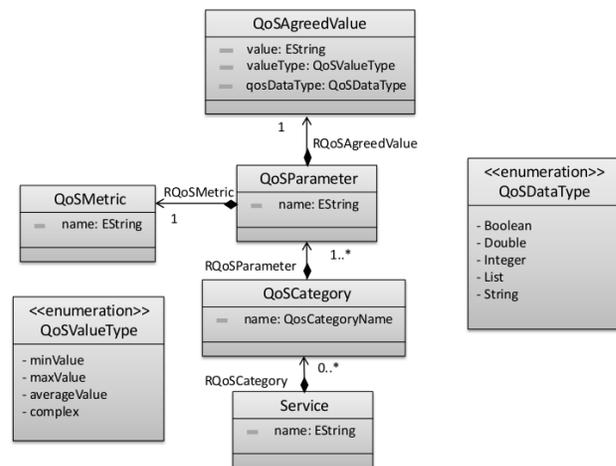


Figure 3. QoS meta-model.

The meta-model enables the QoS profile developer to model QoS attributes for Web services for different business domains. The following changes were made in the meta-model to the meta-model proposed in [8]:

- The `Category` attribute in the `QoSParameter` was modified to include only predefined values specified in the enumeration class `QoSCategoryName`.
- The `QoSLevel` was not considered in this work since the modeled QoS is always fulfilled.

Our meta-model is built using EMF (Core) [9], a modeling framework and code generation facility, which is used to build tools and applications based on a structured data model.

V. QoS PROFILE

QoS profile developer uses QoS meta-model to define and create QoS profiles. A QoS profile is an instance of the meta-model for a specific non-functional business requirement coupled with corresponding QoS criterions in default state or value. All the values of QoS attributes defined by QoS profile developer in QoS profiles are default values that

could be used by Web service developer while applying QoS attributes to the Web service. Web service developer can also change or configure the QoS attributes' values to fit business or project requirements while applying them to the Web service. Hence, QoS profiles are used by Web service developers during development to provide concrete QoS attribute values and apply them to their Web services as policies. Following, we will model three QoS profiles to demonstrate the flexibility of the meta-model. We will present a standardized QoS attribute from the WS-* family and introduce two non-standardized QoS profiles.

The first QoS attribute is from WS-ReliableMessaging. Figure 4 models QoS attributes described as a RM policy assertion example in [6], Section 2.4. In this example, RQoSAgreedValue "60000" for the RQoSParameter InactivityTimeout indicates that if the idle time exceeds 60000 milliseconds, the sequence will be considered as terminated by the service endpoint. RQoSAgreedValue "3000" for RQoSParameter BaseRetransmissionInterval expresses that an unacknowledged message will be transmitted after 3000 milliseconds. RQoSAgreedValue "200" for RQoSParameter AcknowledgementInterval indicates that an acknowledgement could be buffered up to two-tenths of a second by the RM destination.

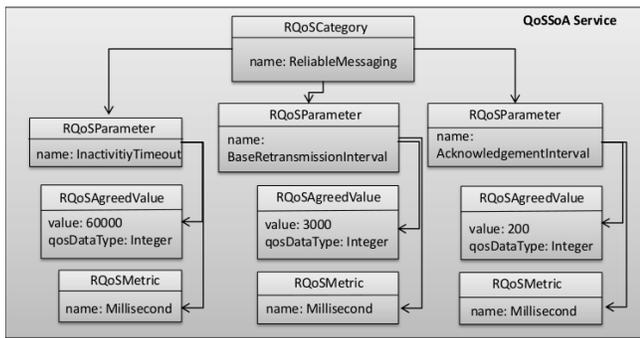


Figure 4. QoS profile for reliable messaging.

The second example models a QoS profile, which is not standardized. A calculator Web service performs arithmetic operations by accepting the operands and the operators. Figure 5 describes calculator Web service constraints for arithmetic operations such as addition and multiplication whose QoS implementation may differ with respect to number overflow. The calculator constraints set minInt and maxInt RQoSParameters for the Web service class or Web service methods. The minInt and maxInt RQoSParameters indicate that all the input and output numbers fall within the range of "0" and "65535". qosDataType in RQoSAgreedValue is set to Integer indicating the data type criterion of the value of RQoSAgreedValue for the calculator constraint.

Figure 6 shows the third example. It is a performance QoS

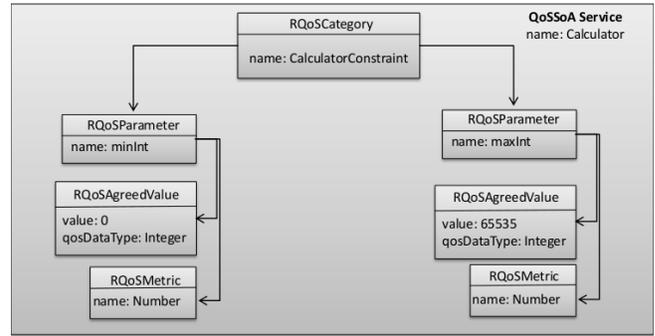


Figure 5. QoS profile for calculator example.

profile where ResponseTime and Throughput are QoS attributes, which are two of the most common used attributes in order to measure performance. Response time refers to the duration, which starts from the moment a request is sent to the time a response is received. Throughput is the maximum amount of requests that the service provider can process in a given period of time without having effect on the performance of the Web service endpoint [10].

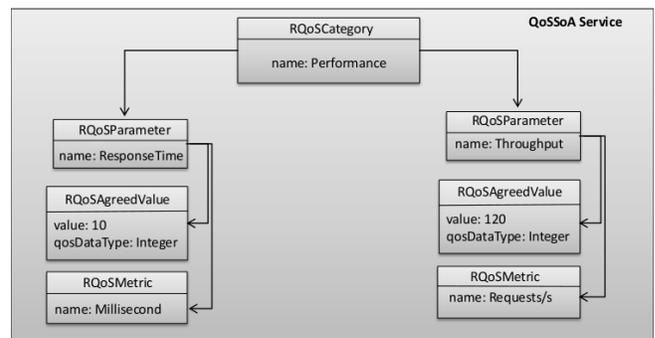


Figure 6. QoS profile for performance.

RQoSAgreedValue "10" for RQoSParameter ResponseTime indicates that the Web service shall guarantee a response within 10 milliseconds where Millisecond is defined as RQoSMetric. Similarly, RQoSParameter Throughput with RQoSAgreedValue "120" indicates that the Web service will be able to handle up to 120 request/second without having any change on the Web service performance. RQoSMetric defines the unit of measure for RQoSAgreedValue.

VI. GRAPHICAL USER INTERFACE

The graphical user interface enables Web service developers to configure or refine QoS profiles with new QoS attributes' values according to the business requirements. It is a graphical tool to associate the QoS values of the modeled QoS attributes.

There are two factors, which decide how the GUI should look like; the first factor is the QoS profile. The QoS profile

specifies the number of categories and the associated QoS attributes with their default values. In our approach, every QoS category is represented by a GUI tab and each tab shows the QoS attributes of the respective QoS category. Such a design ensures a user friendly management and division of QoS attributes based on their categories. For example, if the QoS profile includes three QoS categories performance, reliable messaging and calculator constraints, the QoS profile will be transformed into a GUI, which has three tabs. Each tab will represent a category. If the QoS category, e.g., calculator constraint has two QoS attributes, the QoS category tab on the GUI will represent these two QoS attributes, i.e., `minInt` and `maxInt` with their default values as shown in Figure 7.

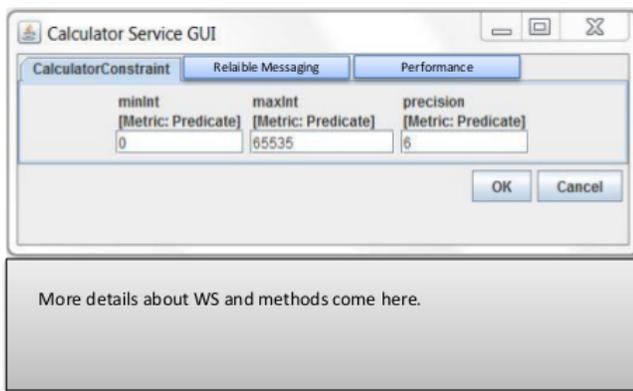


Figure 7. Generated GUI from the QoS profile.

The element `QoSMetric` helps the GUI engine to determine, how the `QoSAgreedValue` shall be presented. For example, if the `QoSMetric` indicates that the QoS attribute is a `Number`, the GUI engine will use a text field for the presentation of this attribute.

The second factor is the Web service endpoint. A list of the Web service methods will be extracted either directly from the Web service endpoint interface (SEI) or from the WSDL. Each extracted method has its own list of QoS attributes. A QoS profile can be associated either to a simple Web service or a set of Web services, i.e., all Web services contained in a WSDL description. If, for example, two Web service methods have two different two different values for “ResponseTime” values, a policy for each method will be created. This will result in creating a separate policy for each selected method. The created policy could also be applied Web service wide. These possibilities give the Web service more flexibility and dynamics.

The QoS profile is transformed into the GUI code using a set of three separate transformation processes. A model-to-model transformation from QoS profile to GUI platform independent model (GUI PIM), which introduces an additional layer of abstraction supporting multiple GUI platforms, a model-to-model transformation from platform

independent GUI model to platform specific GUI model (GUI PSM), for a specific GUI technology such as Swing and SWT and finally a model-to-code transformation from GUI platform specific model to GUI platform specific code, which on execution generates the GUI output. Every transformation is performed based on transformation rules that are defined using transformation languages. The two model-to-model transformations described above are based on the language “Operational Query View Transformation” (QVTO) [11] while the model-to-code transformation is based on “Xpand2” [12].

The transformation from QoS profile to GUI PIM is based on the mappings between QoS profile elements and the GUI elements. The GUI elements are defined in a GUI PIM meta-model, which is used to generate GUI PIM ensuring the extendibility of the solution to other GUI platforms. The GUI PIM meta-model is an abstract GUI model consisting of basic GUI elements. Figure 8 outlines the GUI PIM meta-model. At the root of the GUI PIM meta-model, is a GUI element associated with exactly one `Frame` element in which all the GUI elements are contained. The next level of abstraction of the GUI elements is the `ContainerElement` which can contain `GUIElement`. The `GUIElement` is another abstract form of basic GUI elements divided under four categories, i.e., `ActionElement` (e.g., buttons), `OutputElement` (e.g., labels), `InputElement` (e.g., text fields) and `ChoiceElement` (e.g., list box) [13]. The architecture of the GUI meta-model is similar to the schematics of the QoS meta-model.

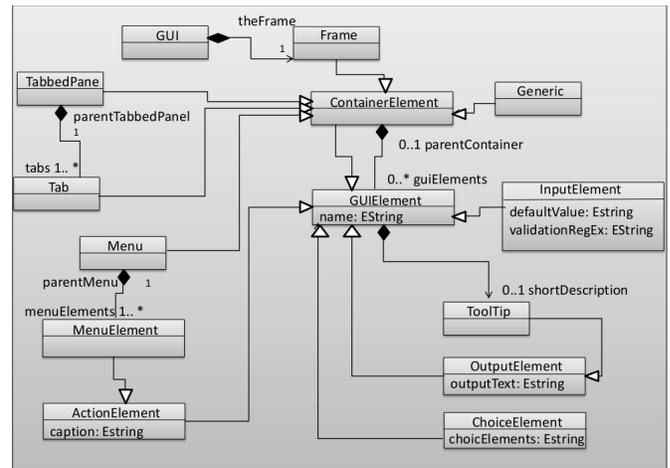


Figure 8. GUI PIM meta-model.

The transformation rules ensure a consistent mapping from different elements of the QoS profile to GUI PIM. Each of the QoS categories from the QoS profile is defined as different tabs in GUI. The quality criterion itself is described with a name, value and a metric representation. Depending on the complexity of the value element, respective mapping is performed, for example, on text field or text area.

The transformation from GUI PIM to GUI PSM is also performed through transformation rules. Essentially, the GUI PSM is platform specific extension of properties of the GUI PIM that contains technology specific entities. The GUI PSM, thus, inherits the elements of GUI PIM and refines them.

The working example shown in Figure 7 is specific to Java Swing framework. The mappings of the elements are performed by simply associating the corresponding elements from the two models. For example, Java Swing element JLabel inherits from the label element of the GUI PIM called Label. Finally from the GUI PSM, the code is generated through a model-to-code transformation. The mappings of each element of the QoS model to GUI PIM and further to Swing specific types are shown in the Figure 9.

QoS Model	GUI PIM Model	Swing Type
Service	Frame	JFrame
QoSCategory	Tab	JTabbedPane
QoSParameter	Generic	JPanel
QoSParameter	Label	JLabel
QoSMetric	Label	JLabel
QoSAgreedValue	TextField or TextArea	JTextField or JTextArea

Figure 9. Mappings from QoS profile to GUI PIM model and corresponding Swing specific types.

The mapping of GUI PSM (Java Swing) into the Java Swing code is straight forward since the elements of GUI PSM bear the same Swing names.

VII. REFINED QoS PROFILE

Web service developer interacts with the generated GUI to refine the QoS attributes' values further according to the business requirements. Hence, the refined QoS profile (see (4) in Figure 2) is an extended version of the QoS profile (see (2) in Figure 2) with well-defined values of the QoS attributes. This component in the solution concept is the first step in building the policy descriptions for the Web services. After entering the values to the QoS attributes in the GUI (see (3) in Figure 2), the refined QoS profile is produced with the set values to QoS attributes. During this process, GUI reads the entered values of QoS attributes. QoS elements in the GUI are searched and matched with the corresponding attributes modeled in the QoS profile. Finally, a refined QoS profile is generated with the entered QoS values assigned to the respective QoS attributes.

Figure 10 shows the refined QoS profile of the calculator service example mentioned earlier in Figure 5 with new values to RQoSParameters minInt and maxInt as

“-32768” and “32767” respectively. The new values to RQoSParameters represent the non-functional requirements set by the Web service developer over the calculator service.

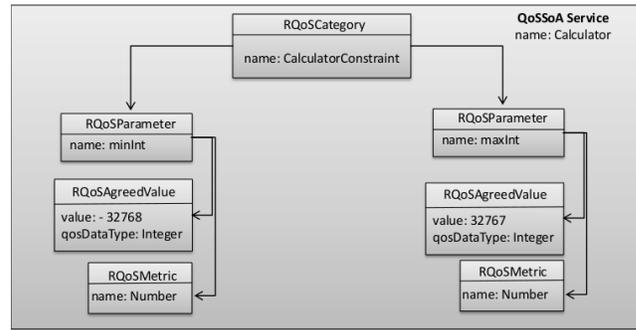


Figure 10. Refined QoS profile for Calculator.

Refined QoS profile is generated by iterating through the elements of GUI PSM, extracting the corresponding elements of QoS profile model with the refined values of QoS attributes (set via the GUI), reframing the quality criteria and finally storing the quality criteria in the refined QoS profile model.

It is of great interest to realize the importance of generating a refined QoS profile component than directly generating the policy model. The refined QoS profile provides the extensibility and flexibility to the generation of policies and applying them to the Web services. It provides an additional layer of abstraction to generate the policy descriptions through policy PIM thereby allowing the solution to extend to multiple policy languages through policy PSM (see Figure 2). Each QoS profile can then extend the Web service with profile specific non-functional quality criteria. This also reduces complexity of integrating multiple profiles into the Web services.

After successful generation of refined QoS profile from the GUI, further transformations to policy model take place. The QoS policy model and the transformations are discussed in the next section.

VIII. QoS POLICY

As a Web service developer refines the QoS attributes on the GUI, all QoS values are assigned to the QoSAgreedValue element in the refined QoS profile. Once the QoS values have been assigned and refined QoS profile is created, a QoS policy model is generated. If, for example, every Web service method has different QoS attributes, a separate policy will be created for every Web service method. A WS-Policy description may include the specifications of more than one QoS attribute depending on the user input in the GUI.

The transformation of the refined QoS profile instance to policy model can also be divided into three different transformation processes. A model-to-model transformation

from refined QoS profile to platform independent policy model (Policy PIM), which introduces an additional layer of abstraction supporting multiple policy languages. Another model-to-model transformation from policy PIM to platform specific policy model (Policy PSM) supporting specific policy languages such as WS-Policy and XACML. And finally a model-to-code transformation from policy PSM to platform specific policy description code. These transformations are performed based on certain transformation rules that are also defined using the transformation languages QVTO and XPand2.

The policy description elements are defined in a policy PIM meta-model which is similar to the GUI PIM meta-model described in Section VI. It is used to generate policy PIM ensuring the extensibility of the solution to other policy formalisms. The policy PIM meta-model is an abstract policy model consisting of four basic policy elements outlined as ServicePolicy, AssertionGroup, Assertion and Property [14]. Figure 11 describes the policy PIM meta-model. ServicePolicy is the root element which can encapsulate any number of AssertionGroup elements as child elements. AssertionGroups can have any number of Assertions as child elements. A Property can be assigned to any of the mentioned elements. A Property extends an element with additional information.

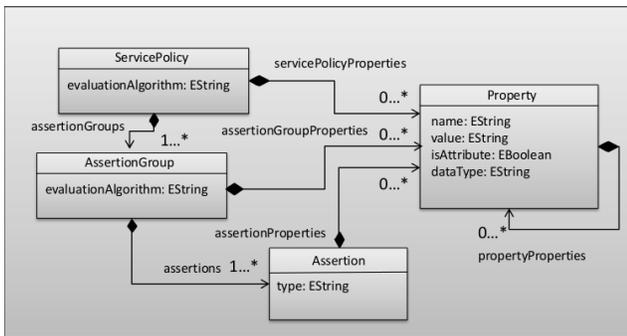


Figure 11. Policy PIM meta-model.

Once the Web service developer refines the QoS profile instance with new QoS values using the GUI, the refined QoS profile to policy PIM transformation process gets executed. This model-to-model transformation is based on transformation rules defining the logical mappings between the elements of refined QoS profile and policy PIM. Figure 12 outlines the mappings.

The transformation from policy PIM to policy PSM is also a model-to-model transformation process. This transformation process can yield multiple policy language specific policy PSMs. Once the policy PSM is created, the policy code is generated via model-to-code transformation XPand2 process.

The following transformation rules are applied to generate policy PIM from refined QoS profile:

Refined QoS Model	Policy PIM Model
Service	ServicePolicy
QoSCategory	-
QoSParameter	Assertion

Figure 12. Refined QoS profile to policy PIM mapping.

- The `Category` attribute in the refined QoS profile declares the name of the policy. For example, the category `ReliableMessaging` is transformed into a `wsm:RMAssertion` element declaring a `ReliableMessaging` policy.
- The element `RQoSParameter` in the refined QoS profile declares a QoS attribute. The `RQoSParameter` element indicates the reliable messaging quality attribute `InactivityTimeout` and is therefore transformed into `wsm:InactivityTimeout` element.
- The element `RQoSMetric` in the refined QoS profile declares a property or how the QoS should be measured. The `Milliseconds` is transformed into `Milliseconds` attribute within the `wsm:InactivityTimeout` element.
- The element `RQoSAgreedValue` in the refined QoS profile declares a QoS value. The value “60000” will be mapped as a value for the QoS attribute.

Listing 1 shows the modeled QoS in Figure 4 after the transformation into a reliable messaging WS-Policy assertion.

```

1 <wsp:Policy wsu:Id="ReliableMessagingPolicy">
2 <wsp:ExactlyOne>
3 <wsp:All>
4 <wsm:InactivityTimeout Milliseconds="60000" />
5 <wsm:BaseRetransmissionInterval
6   Milliseconds="3000" />
7 <wsm:AcknowledgementInterval
8   Milliseconds="200" />
9 </wsp:All>
10 </wsp:ExactlyOne>
11 </wsp:Policy>

```

Listing 1. Reliable messaging WS-Policy description.

Similarly, the WS-Policy descriptions of calculator constraint and performance are generated after their profile transformation into policy assertions. Listing 2 shows the modeled and refined QoS in Figure 10 after the transformation into a calculator constraint WS-Policy assertion.

Listing 3 shows the modeled QoS in Figure 6 after transformation into WS-Policy performance assertion.

Once the policies have been created, policy model (see (4) in Figure 2) will assign the created policies to the Web service endpoint interface. There are several ways to associate a WS-Policy description to a Web service. In our

```

1 <wsp:Policy wsu:Id="CalculatorConstraintPolicy">
2 <wsp:ExactlyOne>
3 <wsp>All>
4   <wscal:minInt Number="-32768"> </wscal:minInt>
5   <wscal:maxInt Number="32767"> </wscal:maxInt>
6 </wsp>All>
7 </wsp:ExactlyOne>
8 </wsp:Policy>

```

Listing 2. Calculator service WS-Policy description.

```

1 <wsp:Policy wsu:Id="PerformancePolicy">
2 <wsp:ExactlyOne>
3 <wsp>All>
4   <wsrm:ResponseTime Milliseconds="10" />
5   <wsrm:Throughput Requests/s="120" />
6 </wsp>All>
7 </wsp:ExactlyOne>
8 </wsp:Policy>

```

Listing 3. Performance WS-Policy description.

proof of concept, we use the CXF policy [15] engine to attach the corresponding policy to either the selected Web service methods or the Web service endpoint interface. CXF uses the @POLICY annotation to signal the compiler that there are policies, which should be considered and assigned to the corresponding Web service while creating its WSDL.

IX. PROOF OF CONCEPT

The solution architecture (Figure 2) discussed in the previous sections can be instantiated in several ways. For our proof of concept we have used a Java based infrastructure. To be precise, the following technologies are used:

- Eclipse IDE - Eclipse Modeling Tools [9].
- QVTO [11] and XPand2 [12] transformation frameworks.
- Java Swing GUI implementation.
- Apache Tomcat application server [16].
- Apache CXF services framework with WS-Policy support [15].

Eclipse Modeling Tools facility is used to build tools and applications based on a structured data model. It provides a pluggable framework to store the model information and by default uses XML Metadata Interchange (XMI) format to preserve the model definition. QoS meta-model in our proof of concept is based on Eclipse Modeling Tools.

WS-Policy [2] provides a policy language to formally describe properties of a behavior of Web services. WS-Policy itself does not come with concrete assertions. Related specifications introduce domain specific assertions, e.g., WS-Security for the security domain. The respective specifications do not only define the syntax, but also the meaning of the assertions and their impact on the Web services runtime behavior.

Operational Query View Transformation (QVTO) is used to create transformation rules for model-to-model transformations. It is the sublanguage of Query View Transformation (QVT) [17].

XPand2 is used to perform model-to-code transformations. XPand2 is administered in Eclipse Modeling Project (EMP) but has its roots from OpenArchitectureWare Framework. In our proof of concept, XPand2 is used to generate a Swing based GUI.

Apache CXF is an open source services framework. Apache CXF helps in building and developing services using frontend programming APIs, like JAX-WS and JAX-RS. Apache CXF includes a broad feature set, but it is primarily focused on supporting Web service standards including WS-Policy and frontends.

The main purpose of Apache CXF in our example is WS-Policy support it offers and its possibility to associate WSDLs with existing WS-Policies through annotations.

This section uses the calculator example mentioned in this paper to attach quality attributes and to demonstrate the overall working of the solution.

We begin with Eclipse Modeling Tools and define the QoS meta-model. QoS profile developer uses and instantiates QoS meta-model to provide QoS profile. QoS profile for a simple calculator is generated with two quality attributes, i.e., minInt and maxInt. The QoS profile, then, undergoes two automatic QVTO transformations, i.e., QoS profile to GUI PIM, and GUI PIM to GUI PSM. Listing 4 shows a sample transformation rule of QoS profile to GUI PIM.

```

1 modeltype QOS 'strict' uses
2   QoSSOAMetaModel('http://qosmetamodel/1.0');
3 modeltype GUI_PIM 'strict' uses
4   guipimmetamodel('http://guipimmetamodel/1.0');
5 transformation QoS2GUIPIMTransformation
6   (in qos : QOS, out guiPim : GUI_PIM);
7
8 property validationRegExDouble = '\\d+{\\.|,}\\d+';
9 property validationRegExInteger = '\\d+';
10
11 main() {
12   qos.objects()[Service]->xmap serviceToGUI();
13 }
14
15 mapping Service::serviceToGUI() : GUI
16 when {
17   self.RQoSCategory->size() > 0;
18 }
19 {
20   theFrame := object Frame {
21     name := self.name + ' GUI';
22   };
23   ...
24   ...
25   ...

```

Listing 4. Sample .qvto code for QoS profile to GUI PIM transformation.

The participating models are defined by using the keyword `modeltype`. Keyword `transformation` describes the source and the target model. The mapping functions are defined and called from within the `main()` function, which is called on the execution of the transformation file. Lines 20-21, for example, add `Frame` to the GUI. GUI PSM in our proof of concept is a GUI implementation, which is generated via transformation rules specific to the Java Swing environment from the abstract GUI PIM. The transformation

follows the similar syntax as described for QoS profile to GUI PIM transformation. Listing 5 shows the sample transformation.

Once the GUI PSM is generated, the code is created automatically via model-to-code Xpand2 transformation process. Xpand2 template file reads the elements of the GUI PSM and translates them into Java Swing elements' code. On execution, it presents the Web service developer with the GUI. Figure 7 shows the generated GUI for our proof of concept calculator example with default values. Web service developer can interact with the GUI and insert new values to the quality criterion of calculator example.

```

1 modeltype GUI_PIM 'strict' uses
2  guipimmetamodel('http://guipimmetamodel/1.0');
3 modeltype GUI_PSM 'strict' uses
4  guipmswingmetamodel('http://guipmswingmetamodel/1.0');
5 transformation GUIPIM2GUIPSMTransformation
6  (in pim : GUI_PIM, out psm : GUI_PSM);
7
8 main() {
9  pim.objects()[GUI]->map guiToGUI();
10 }
11
12 mapping GUI::guiToGUI() : GUI {
13  theFrame := object JFrame {
14    name := self.theFrame.name;
15  };
16  theFrame.guiElements +=
17  self.theFrame.guiElements->select
18  (elem | elem.ocIsTypeOf(TabbedPane))
19  .oclAsType(TabbedPane)
20  ->map tabbedPaneToJTabbedPane();
21
22  theFrame.guiElements +=
23  self.theFrame.guiElements->
24  select(oclIsTypeOf(Generic)).oclAsType(Generic)->
25  map genericToJPanel();
26  ...
27  ...
28  ...

```

Listing 5. Sample .qvtoc code for GUI PIM to Java Swing transformation.

Each generic element from the GUI PIM is mapped with respective Java Swing GUI element. Lines 16-20 show the mappings of GUI PIM element `TabbedPane` to the corresponding GUI PSM (Java Swing) element `JTabbedPane`. Figure 9 shows the mappings of the elements.

The process of generating refined QoS profile is similar to the process of instantiation of QoS meta-model described before to generate QoS profile.

The refined QoS profile undergoes two model-to-model QVTO transformations, i.e., refined QoS profile to policy PIM and policy PIM to policy PSM as well as a model-to-code Xpand2 transformation from policy PSM to get WS-Policy code.

Refined QoS profile to policy PIM transformation is similar to the transformation described above, i.e., QoS profile to GUI PIM. Listing 6 shows the sample transformation. Lines 19-21 maps the `QoSParameters` of a `QoSCategory` to respective assertions.

Policy PSM in our proof of concept is WS-Policy specification. It is generated from the abstract Policy PIM via

```

1 modeltype QoS 'strict' uses
2  QoSMetaModel('http://qosmetamodel/1.0');
3 modeltype PIM 'strict' uses
4  "http://webuser.hs-furtwangen.de/~passfall/PIM";
5
6 transformation QoS2PolicyModelTransformation
7  (in qos : QoS, out policy : PIM);
8
9 main() {
10  qos.objects()[Service]->
11    xmap Service2ServicePolicy();
12 }
13
14 mapping Service::Service2ServicePolicy() : ServicePolicy
15 {
16  evaluationAlgorithm := self.name + ' Policy';
17  var assertionGroupElement :=
18  object AssertionGroup {};
19  assertionGroupElement.assertions +=
20  self.RQoSParameter.RQoSParameter ->
21  xmap Parameter2Assertion();
22  assertionGroups += assertionGroupElement;
23  ...
24  ...
25  ...

```

Listing 6. Sample .qvtoc code for refined QoS profile to policy PIM transformation.

```

1 modeltype PIM "strict" uses
2  "http://webuser.hs-furtwangen.de/~passfall/PIM";
3 modeltype WSP "strict" uses
4  "http://webuser.hs-furtwangen.de/~passfall/PSMWSPolicy";
5 transformation PIM2WSP(in Source: PIM, out Target: WSP);
6
7 helper findNamespaces
8  (param : Set(PIM::Assertion)) : List(WSP::Namespace) {
9
10 var nss : List(WSP::Namespace);
11 param->switch(s) {
12 case (s.type = "SignedElementsIndicator") {
13   var ns := object Namespace {
14     prefix := "sp";
15     url :=
16     "http://docs.oasis-open.org/ws-sx/
17     ws-securitypolicy/200702";
18   };
19   ...
20   ...
21   ...
22 }
23 }
24 main() {
25  Source.objects()
26  [ServicePolicy]->xmap ServicePolicyToPolicy();
27 }
28
29 mapping ServicePolicy::ServicePolicyToPolicy() : Policy {
30  var assertions := Source.objectsOfType(PIM::Assertion);
31  namespaces := findNamespaces(assertions);
32  alternatives += self.assertionGroups->
33  xmap AssertionGroupToAlternative();
34  ...
35  ...
36  ...

```

Listing 7. Sample .qvtoc code for policy PIM to WS-Policy transformation.

transformation rules specific to the WS-Policy specifications. Listing 7 shows the sample transformation. Each generic element from the Policy PIM is mapped with a respective WS-Policy specific element. Line 26 is the function call to `ServicePolicyToPolicy` function. Line 32 performs the mapping of `AssertionGroup` to WS-Policy `Alternative`.

Figure 13 shows the mappings of the elements.

Policy PIM Model	Policy PSM Model (WS-Policy)
ServicePolicy	Policy
AssertionGroup	Alternative
Assertion	Assertion
Property	AssertionProperty

Figure 13. Policy PIM to WS-Policy mappings.

Once the policy model for WS-Policy is generated, the application of the policies is attached to the Web service. Now, Apache CXF policy engine attaches the corresponding policy to either the selected Web service method or the Web service endpoint interface. Apache CXF uses the @POLICY annotation to signal the compiler that there are policies, which should be considered and assigned to the corresponding Web service while creating the Web service WSDL.

X. RELATED WORK

In our research for related work, an approach, which nearly investigates our approach or even a part of it was not found. Most of the recent works on QoS-aware Web services focus on QoS-aware Web services compositions. They investigate methods, algorithm or frameworks in order to better compose Web services according to their QoS attribute. Such works could be found in [18]–[20]. In this section, we will describe papers, which propose either QoS meta-models or policy editors.

Tondello et al. [21] proposes a QoS-Modeling ontology, which allows QoS requirements to be specified in order to fully describe a Web service in terms of quality. However, this proposal focuses on using QoS specification for semantic Web service descriptions and Web service search. This approach, however, contains many variables and many characteristics in ontology for semantic Web services, which does not flow in the same direction as this work intends to.

Suleiman et al. [22] addresses the problem with Web service management policies during design. The authors presented a solution, which uses a novel mechanism. It generates WS-Policy4MASC policies from corresponding UML profiles semi-automatically and feedback information monitored by the MASC middleware into a set of UML diagram annotations.

D'Ambrogio [23] introduced a WSDL extension for describing the QoS of a Web service. It uses a meta-model transformation according to the MDA standard. The WSDL meta-model is extended and transformed into a new WSDL model called Q-WSDL, which supports QoS descriptions. As D'Ambrogio favour an approach, which does not support

introducing a new additional language on top of WSDL, our approach uses standards for the description of QoS attributes in Web services.

WSO2 WS-Policy editor [24] offers an integrated WS-Policy editor with the WSO2 application server. The editor offer two policy views: a source view and a design view. The source view shows the policy in its XML format and the design view shows the policy as a tree view. The user will be able to add and remove element to and from the policy. However, this policy editor only offers support for WS-Security and WS-ReliableMessaging. A support for new QoS attributes is not mentioned.

NetBeans [25] offers a graphical tool, which allows users to graphically configure security and reliable messaging to a Web service. Extending this tool, however, is complex due to the lack of documentation and its dependability to NetBeans API and Glassfish [26].

All these works discuss QoS attributes after the Web services are developed. Our approach offers a solution to develop a QoS-aware Web service.

XI. FUTURE WORK

In [27], we presented the design of a comprehensive tool chain that facilitates development, deployment and testing of QoS-aware Web services. This paper is a part of the work presented in the tool chain, which elaborates a concept for managing quality of service attributes for Web services. Future works will include different tasks, which will be individually explained in this section.

In Section VI, we introduced a GUI, which is dynamically generated depending on the QoS attributes modeled in the QoS profile described in Section V. However, the generation of the GUI is platform-specific. This GUI is only a proof of concept in order to demonstrate the feasibility of this approach. Our goal is to create a GUI using MDA as a base for our approach, which will allow the dynamic GUI generation to different platform.

Section VII indicates that the QoS profile will be transformed to a QoS policy. In this paper, we have only considered WS-Policy as a policy language in order to prove that the concept really works. It is the intention of this approach to offer QoS model transformation support to more than one policy language. This will increase the flexibility of our approach.

In [28], we offered a solution architecture, which collects real time data about applied QoS attributes from the SOA environment: the purpose of this architecture is to evaluate the compliance of the entire SOA with the QoS attributes described in the SOA QoS policy. It is our intention to use the meta-model mentioned in Section IV for the evaluation and monitoring of the SOA environment.

This paper presents an approach of how QoS attributes could be easily modeled and transformed into an adequate policy language. However, a policy without a handler, which

enforces the policy on the Web service, is only half the solution. Future works include a repository component, which is designed to store QoS handlers. This repository will include handler implementation, handler configurations, and test cases.

XII. CONCLUSION

The design and implementation of an SOA that contains QoS attributes is difficult. There are tools and IDEs, which help developers to ease the process of creating programs, minimizes their error rates, designing and implementing such complex systems. But, to create a QoS policy and conjugate it with a Web service still requires a good knowledge of its grammar and its mechanism. It is highly desirable to have tools, which help developers to model QoS attributes, simplify the configuration and automate applying QoS attributes to Web services. First steps towards such tools have been made, but the overall support for developers needs to be highly improved.

In this paper we presented a tool chain, which increases the support for two developer roles: QoS profile developer and Web service developer. The former, by the use of a generic QoS meta-model, defines QoS profiles for targeted QoS attributes. The approach thereby support the modeling of new QoS attributes. From a QoS profile a corresponding GUI is generated, which supports the Web service developer to refine the QoS profile, to generate a policy model, and to apply the corresponding policies to Web services. Implementation details, like the usage of WS-Policy and adding necessary annotation in source code, are hidden from the Web service developer.

Throughout the paper we described the necessary modifications, user interactions and transformations step by step. At the end, the feasibility of the approach was shown by a proof of concept.

In summary, the approach is a major step towards an increased support for constructing QoS-aware Web services. It eases and unifies the development process and helps to reduce the error rate, development effort and the overall costs.

XIII. ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for giving us helpful comments.

This work has been supported by the German Ministry of Education and Research (BMBF) under research contract 017N0709.

REFERENCES

- [1] A. Al-Moayed and B. Hollunder, "An approach to model, configure and apply QoS attributes to web services," in *Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA 2011)*. Xpert Publishing Services, 2011, pp. 405–410.
- [2] W3C, "Web Services Policy Framework - Version 1.5," September 2007, last access: 20.12.2012. [Online]. Available: <http://www.w3.org/TR/ws-policy/>.
- [3] T. Erl, A. Karmarka, P. Walmsley, H. Haas, U. Yalcinalp, C. K. Liu, D. Orchard, A. Tost, and J. Pasley, *Web Service contract Design & Versioning for SOA*. Prentice Hall, 2009.
- [4] B. Hollunder, M. Hüller, and A. Schäfer, "A methodology for constructing ws-policy assertions," in *Proceedings of the 2nd International Conference on Engineering and Meta-Engineering (ICEME 2011)*, 2011, pp. 112–117.
- [5] OASIS, "Web Services Security Policy - Version 1.3," April 2009, last access: 20.12.2012. [Online]. Available: <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/>.
- [6] OASIS, "Web Services Reliable Messaging Policy - Version 1.2," February 2009, last access: 20.12.2012. [Online]. Available: <http://docs.oasis-open.org/ws-rx/wsrml/v1.2/wsrml.pdf>.
- [7] OASIS, "eXtensible Access Control Markup Language (XACML) Version 2.0," OASIS, February 2005, last access: 20.12.2012. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.
- [8] D. Malfatti, "A Meta-Model for QoS-Aware Service Compositions," Master's thesis, University of Trento, Italy, 2007.
- [9] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, E. Gamma, L. Nackman, and J. Wiegand, Eds. Addison-Wesley Professional, 2008.
- [10] E. Kim, Y. Lee, Y. Kim, H. Park, J. Kim, B. Moon, J. Yun, and G. Kang, "Web service quality factors version 1.0," OASIS, Tech. Rep., 2011, last access: 20.12.2012. [Online]. Available: <http://docs.oasis-open.org/ws-qm/WS-Quality-Factors/v1.0/WS-Quality-Factors-v1.0.pdf>.
- [11] R. Dvorak, "Model Transformation with Operational QVT," Borland Software Corporation, 2008, <http://www.eclipse.org/m2m/qvto/doc/M2M-QVTO.pdf>, last access: 20.12.2012. [Online]. Available: <http://www.eclipse.org/m2m/qvto/doc/M2M-QVTO.pdf>.
- [12] E. Galileo, "Xpand/ Xtend/ Check Reference," The Eclipse Foundation, last access: 20.12.2012. [Online]. Available: <http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.xpand.doc/help/ch01s06.html>.
- [13] M. Hermann and A. Hülzenbecher, "M2M-Transformation zur Generierung einer grafischen Benutzeroberfläche in einem QoS-SOA Kontext," Hochschule Furtwangen University, 2011, informatik Journal, Faculty of Informatics.
- [14] A. Passfall, T. Rübsamen, and R. Teckelmann, "Modellbasierte Erzeugung von Policy-Dokumenten," Hochschule Furtwangen University, 2011, informatik Journal, Faculty of Informatics.
- [15] "Apache CXF: An Open-Source Services Framework," The Apache Software Foundation, last access: 20.12.2012. [Online]. Available: <http://cxf.apache.org/>.

- [16] “Apache Tomcat.” The Apache Software Foundation, last access: 20.12.2012. [Online]. Available: <http://tomcat.apache.org/>.
- [17] OMG, “Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification,” last access: 20.12.2012. [Online]. Available: <http://www.omg.org/spec/QVT/1.0/>.
- [18] M. H. Agdam and S. Yousefi, “A Flexible and Scalable Framework For QoS-aware Web Services Composition,” in *Proc. 5th Int Telecommunications (IST) Symp*, 2010, pp. 521–526.
- [19] P. Bartalos and M. Bielikova, “QoS Aware Semantic Web Service Composition Approach Considering Pre/Postconditions,” in *Proc. IEEE Int Web Services (ICWS) Conf*, 2010, pp. 345–352.
- [20] H. Kil and W. Nam, “Anytime Algorithm for QoS Web Service Composition,” in *Proceedings of the 20th international conference companion on World wide web*, ser. WWW '11. New York, NY, USA: ACM, 2011, pp. 71–72, last access: 20.12.2012. [Online]. Available: <http://doi.acm.org/10.1145/1963192.1963229>.
- [21] G. Tondello and F. Siqueira, “The QoS-MO Ontology For Semantic QoS Modeling,” in *Proceedings of the 2008 ACM symposium on Applied computing*, ser. SAC '08. New York, NY, USA: ACM, 2008, pp. 2336–2340, last access: 20.12.2012. [Online]. Available: <http://doi.acm.org/10.1145/1363686.1364239>.
- [22] B. Suleiman and V. Tasic, “Integration of UML Modeling and Policy-Driven Management of Web Service Systems,” in *Proc. ICSE Workshop Principles of Engineering Service Oriented Systems PESOS 2009*, 2009, pp. 75–82.
- [23] A. D’Ambrogio, “A Model-driven WSDL Extension for Describing the QoS of Web Services,” in *Web Services, 2006. ICWS '06. International Conference on*, sept. 2006, pp. 789–796.
- [24] WSO2, “WSO2 WSAS: The WS-Policy Editor 3.2.0 - User Guide,” WSO2, April 2010, last access: 20.12.2012.
- [25] “NetBeans IDE,” Oracle Corporation, last access: 20.12.2012. [Online]. Available: <http://netbeans.org>.
- [26] “Glassfish Application Server,” last access: 20.12.2012. [Online]. Available: <http://glassfish.java.net>.
- [27] B. Hollunder, A. Al-Moayed, and A. Wahl, *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. IGI Global, 2011, ch. A Tool Chain for Constructing QoS-aware Web Services, pp. 172–188.
- [28] A. Wahl, A. Al-Moayed, and B. Hollunder, “An Architecture to Measure QoS Compliance in SOA Infrastructures,” in *Proceedings of the Second International Conferences on Advanced Service*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 27–33.

Testing Object-Oriented Code Through a Specifications-Based Mutation Engine

Pantelis Stylianos Yiasemis

Department of Computer Engineering and Informatics,
Cyprus University of Technology
Limassol, Cyprus
email: pantelis.yiasemis@cut.ac.cy

Andreas S. Andreou

Department of Computer Engineering and Informatics,
Cyprus University of Technology
Limassol, Cyprus
email: andreas.andreou@cut.ac.cy

Abstract—This paper presents a simple, yet efficient and effective mutation engine that can produce mutations of object-oriented source code written in the C# and Visual Basic languages as an extension of previous work on the topic [1]. The engine produces mutants based on user selected mutation operators the number of which is bounded by the specifications declared in the source code with the aid of Code Contracts. The specifications are described using a set of pre- and post-conditions and invariants. The engine consists of four distinct and integrated components; a syntactic verification component, a static analysis component, a mutation generation component, and a test case quality assessment component. A series of experiments are conducted which show that the proposed engine is able to locate a fault and efficiently propose the proper correction. In addition, the scalability of the proposed approach is assessed in terms of time and performance with respect to different program sizes.

Keywords—mutation testing; mutation engine; specifications;

I. INTRODUCTION

The rapid evolution of technology has lead to the creation of a large variety of tools that automate a number of activities within the process of software development. Computing power increases in almost exponential rates, a fact that supports the development of better and faster software systems, which, in turn, exercises pressure on their reliability as typically these systems become increasingly more complex. The competition that exists between software development companies pushes them to increase their productivity by developing the software in tighter time frames having a direct effect on the quality of the software developed.

Software Testing efficiency, or better the lack of it, is one of the most important reasons for inadequate quality control in today's software development. Software testing is a way of making sure that a software system meets its specifications while being correct and appropriate ([2], [3]). Software testing is a quite complex process that needs to be correctly performed; it thus consumes a large percentage of the time and budget of the whole development process. In some occasions it even surpasses the time and budget needed for the creation of the software product [4]. Its main purpose is the improvement of the functional behavior of a system under development, by revealing and locating faults in source code.

The software testing process comprises two main activities, the correct identification of faults and their correction (debugging). Faults can be incorrect steps or data definitions in a program that when executed together lead to

failure. Such faults are also called errors, anomalies, inconsistencies or bugs [5]. Identifying faults is more time consuming than correcting faults in software testing. This leads to the conclusion that there is a constant need for developing tools that will aid the acceleration, correctness and automation of the testing process, by guiding developers to locate and correct faults more efficiently and effectively.

The aim of this paper is to introduce a mutation engine for source code written in two popular object-oriented programming languages, namely C# and VB. Mutations are replacements of code statements performed through certain operators that correspond to specific types of errors. These replacements produce the so-called mutant programs which are then used in order to assess the quality of a test case set as regards to its ability to identify faults in code. The proposed engine constitutes the backbone of a novel mutation testing technique that takes into consideration the specifications of the program for creating only valid mutants. The engine is implemented in Visual Studio 2010 and consists of four components: The first offers the ability to validate the grammatical correctness of the source code; the second provides a form of static analysis for exporting useful information that can be used to process/modify the source code; the third involves the production of mutations of the original source code, while the fourth facilitates the identification of faults, as well as the assessment of the quality of test data.

This work constitutes an extension of previous work on the topic [1], which introduced a Mutation Engine for C# making use of Code Contracts to limit the number of produced mutants thus decreasing the time needed for fault localization. The new ground investigated in this paper may be summarised to the following: (a) The Mutation Engine has been extended to work with code written not only in C# but in VB as well. (b) The assessment of the resulted mutants and the correct identification and correction of faults for a series of examples are performed for both programming languages with comparisons between their results. (c) The engine was tested against larger versions of code and the time performance for locating the faults and producing all the mutants was assessed for multiple program sizes, with the lines of source code being increased by two, four, six and eight times respectively.

The rest of the paper is structured as follows: Section II describes briefly the basic concepts that form the necessary technical background of this work. Section III presents the mutation engine, its architecture and key elements ruling the

generation of mutations, along with a brief demonstration of the supporting software tool. Section IV describes a set of experiments and the corresponding results that indicate the correctness and efficiency of the proposed approach. Finally, Section V concludes the paper and suggests some steps for future work.

II. TECHNICAL BACKGROUND

According to Khan [6], there are three kinds of software testing techniques. These are White Box Testing (WBT), Black Box Testing (BBT) and the mixing of the two called Gray Box Testing (GBT). Each techniques offers its own advantages and disadvantages, differing in the way test cases are created and executed. In BBT the test cases are created based on the functions and specifications of the system under testing without the need for actual knowledge of the source code. WBT requires that the tester has full access to the source code and knows exactly the way it works. An advantage of this method is the ability to locate coincidental correctness, that is, the case where the final result is correct but the way it is calculated is not. Furthermore, all possible paths of code execution may potentially be tested offering the means to identify errors or/and locate parts of dead code, that is, parts that are actually never executed. GBT combines the testing methodology of WBT and BBT, meaning that it tests a system against the specifications defined but also it uses information from the source code to create the test cases. It needs more knowledge of the internals of a system than BBT but less than WBT.

Different techniques have been proposed for WBT [6] making use of the structure of the source code or the sequence of execution, giving birth to static code analysis for the former and dynamic testing for the latter. This paper concentrates on dynamic testing where the actual flow of execution drives test data production. One such technique that has gained serious interest among the research community is Mutation Testing (MT).

Various research studies propose Mutation Testing as the basic element of their approach to software testing (e.g., [7],[8]). MT is a relatively new technique introduced by DeMillo et al. [9] and Hamlet [10], which is based on the replacement of code statements through certain operators that correspond to specific types of errors, producing the so-called mutant programs. These programs are then used to assist in producing or/and assessing the quality of test data as regards revealing the errors in the mutants [11].

The general idea behind MT is that the faults being injected correspond to common errors made by programmers. The mutants are slightly altered versions of programs which are very close to their correct form. Each fault is actually a single change of the initial version of the program. The quality of a produced set of test cases is assessed by executing all the mutants and checking whether the injected faults have been detected by the set or not. This assessment is based on a Mutation Score (MS), which is the ratio of “killed” mutants against the non-equivalent mutants. The purpose of mutation analysis is to aid in creating a test case set of high quality, that is, a set able to produce a MS closer to 1. Such a test can be used to detect all the faults that may exist in the code.

It is possible to produce a large number of variations of a program and the faults that may contain, thus traditional MT targets only groups of faults that are closer to the original version of the code. This practice is based on the Competent Programmer Hypothesis (CPH) and the Coupling Effect (CE). The CPH states that the code written by programmers is almost correct. CE states that when identifying simple faults with a set of test data, the same test data can also identify larger and more complex faults [12]. While recent work in the field of MT deals with high order mutations [11] [13], this paper targets only on first order mutants (simple mutants) as these may be considered good enough, based on CPH and CE, for performing adequate testing of program code. Complex faults are represented by complex mutations consisting of more than one change in the code, whilst simple faults are represented by a single mutation (syntactic change) to a program.

There are a number of ways to represent program code. Each provides a particular way to understand a program and manage its source code. Most of them use graphs or/and binary trees that are able to depict graphically how the program actually works. The Control Flow Graph (CFG) is one such way of graphically representing the possible execution paths. Each of the nodes in a CFG corresponds to a single line of program source code. The arcs connecting nodes represent the flow of execution. A CFG may be used as the cornerstone of static analysis, where its construction and traversing offers the ability to identify and store information about the type of statements present in the source code and the details concerning the alternative courses of execution. A fine example is the BPAS framework introduced by Sofokleous and Andreou [14] for automatically testing Java programs. A CFG may also drive the generation of test data by providing the means to construct an objective function for optimization algorithms to satisfy (algorithms by evolution such as the one proposed in Michael et al. [15]).

The Visual Studio (VS) platform [16] has been constantly evolving becoming one of the most widely used platforms today in the software industry. This is partly due to the fact that it provides the ability to create a number of different types of applications, like window-apps, web-apps, services, classes etc. The wide acceptance of VS has driven the development of a number of third party tools and plug-ins that enhance the platform with even more functionality, making development of special-purpose applications simpler and easier. The aforementioned advantages of VS2010 suggest that its use might be quite beneficiary for software testing, and more specifically for developing a new mutation testing tool.

Code Contracts (CC) were introduced by VS2010 as a means to encode specifications [17], but can be installed on other versions of Visual Studio as well. CC may consist of pre-conditions, post-conditions and invariants. The aim is to improve the testing process through both runtime and static checking. Runtime checking takes place while the program executes and produces an exception when the specifications in the code are not met. Static contract verification is performed while the project is under development. It produces a warning when a condition is not satisfied and also proposes a solution to fix the relevant code. CC also assist in documentation

generation by producing an XML file with information from the CC. CC can be used on any .Net platform that contains the Contracts class, or, if building a project on a platform that does not support CC (e.g., older versions of Silverlight, Windows Phone 7, etc), a reference to the assembly Microsoft.Contracts.dll should be added to the project.

Code Contracts were developed from knowledge obtained from the Spec# programming system, an attempt made by Microsoft to provide a way for more cost effective and higher quality software. SPEC# is in essence a formal language for API contracts that permits specification and reasoning of object invariants, even in parallel processing environments or when callbacks exist in the code. Using a CC enables a programmer to create a detailed set of specifications that will be used to verify a program with the use of the static program verifier. The latter checks if a program satisfies the specifications with no runtime errors. SPEC# is being developed as a research project by Microsoft Research's Programming Languages and methods group [18].

The mutation engine introduced in this paper is partly based on the aforementioned concepts. More specifically, it utilizes CFG and static analysis as in [14] to extract the information needed for analyzing and describing adequately the source code under investigation. Moreover, it employs CC to embed the specifications required in order to assess whether a program functions properly. The mutation engine utilizes runtime checking to limit the production of meaningful mutant programs, that is, programs that do not violate their original specifications. Lastly, it incorporates an automatic test case assessment module that either evaluates the quality of a given test case set, or identifies faults in the original code, and proposes the proper correction that also satisfies the specifications. The engine offers a means for both the automation of software testing and a reduction in the time required for software testing.

III. MUTATION ENGINE

The mutation engine is implemented in the VS2010 platform. VS2010 was selected partly because it is a relatively newly introduced platform, meaning that the components developed may be used as the backbone for future tools and studies based on this platform, without facing any incompatibility issues compared to the use of older platforms. Also, to the best of our knowledge, at present no other such system exists. The engine was originally designed to work with the C# programming language. A number of additions and enhancements were introduced to the engine for supporting Visual Basic as an extension to our previous work [5].

A. Research Strategy

The first step to design and implement the mutation engine was the selection of particular technologies to be incorporated in the proposed tool. Therefore, CFG were employed to aid in the static analysis and CC were chosen to limit the number of produced mutants.

CFG were chosen to represent the code as they give information about the flow of execution, but most importantly they identify and store information about the

type of statements present in the code and additional details regarding alternative courses of execution. This information can then be used for static analysis of the code, which is a preprocessing stage that enables the gathering of critical information as regards specific parameters of the program under testing. This assists in the application of the mutation operators on the source code as it identifies everything present in the code (variables, classes, statements) and the proper mutations can then be applied to each of the elements identified. CC were selected as a means to describe the conditions that exist in the specifications of the program in order to help eliminate those mutations that do not satisfy the specifications.

B. Architecture

The architecture of the proposed mutation engine is depicted graphically in Figure 1 where four major components enable the execution of the engine's stages.

1) Syntactic verification component

The first is a source code validation component, which compiles the source code and presents any erroneous lines. This component takes as input a source code file (.cs or .vb), or an executable file (.exe), or a dynamic link library file (.dll), as well as the project file (.csproj or .vbproj). The project file provides the validation component with information for references in libraries and files that the source code uses and are part of the program. Validation includes compiling the source code and making sure that no syntactic or other compilation errors exist so as to proceed with the second stage of the engine which is the production of mutations. Otherwise the engine terminates.

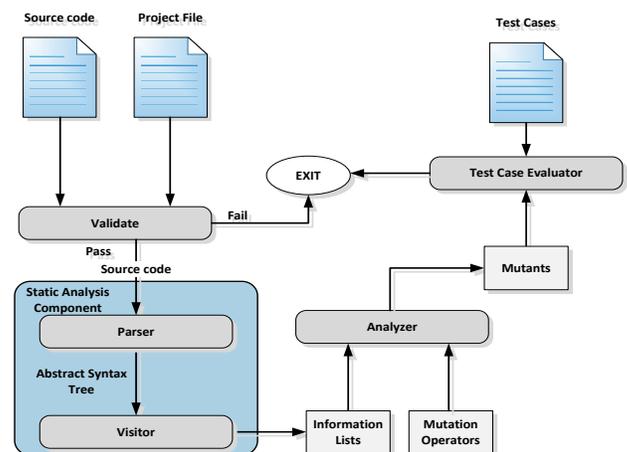


Figure 1. The mutation engine architecture

2) Static analysis component

The second component performs static analysis of the source code without the need of an executable form of the program under testing. Static analysis is the extraction of useful information from the source code concerning the structure of the program. This component takes as input the source code file and uses the class `AbstractSourceTree` (AST) of SharpDevelop [19] to model the abstract syntax tree of the code. While compiling a source code file, a binary tree (the

AST) is created, each node of which represents a line of code. Traversing this binary tree, offers access to any part of the source code.

The static analysis component described above consists of two sub-components, the *Parser* and the *Visitor*. The *Parser* analyses the source code and creates the AST as mentioned earlier. The *Visitor* passes through the AST collecting useful information, while giving the opportunity for the user to make changes and additions to the information stored. The implementation of the *Visitor* utilised the *AbstractAstVisitor* class of SharpDevelop, with some minor additions to help accessing all the nodes of the AST, both at the high (classes and their parameters, inheritance, etc.) and the low level (assignments, conditional statements, unary statements, etc.) characteristics of the programming language. The *Visitor* recursively visits each node and stores in stack-form lists all the information identified according to the node's type. In the experiments described in the next section thirteen such lists were created; nevertheless, the way the *Visitor* is structured enables the addition of any new lists or the modification of existing ones in a quite easy and straightforward manner.

3) Mutation generation component

The third component is the heart of the mutation engine. This component analyses the information stored in the lists created by the *Visitor* so as to identify the structure and content of the source code, and creates mutated programs by applying a number of predefined operators to the initial program. These mutators are responsible for creating a number of different variations of the initial source code. Each mutation is based on one or more grammatical rules that do not breach the grammatical correctness of the resulting program.

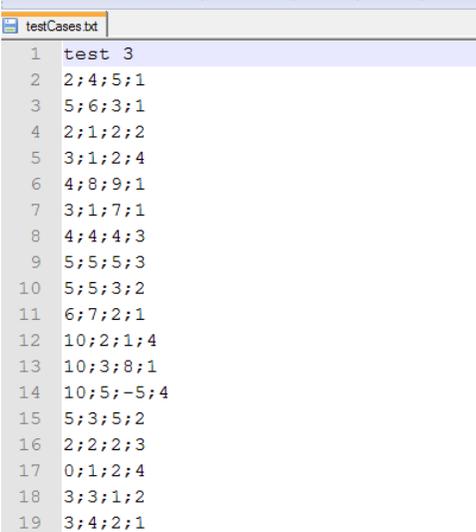
4) Test case quality assessment component

The final component of the mutation engine is the automatic test case quality assessment and fault detection component. It takes as input a text file containing the test cases. The test case file includes a header containing the name of the function to be called and the number of arguments that are needed as input, while the rest of the file contains the test cases values and the expected results for each set of inputs. Figure 2 presents an example of a test case file. The function that is going to be tested is called `test` and takes 3 parameters, as seen in the header of the file. The rest of the file contains, for each test case, the input values and expected results separated by semi-columns. The test case evaluator then loads all the test cases found in the test case input file and applies them to the original program.

If the program returns the expected values for each of the test cases, the tool continues applying each mutation to the original code and evaluating the results of each test case.

When all the test cases have been applied to all the mutants, the engine calculates the Mutation Score, along with information about which mutants were detected, which were not, the mutants that could not be compiled and some other run-time information. In the case where the initial source code fails to give the expected output for a specific test case, the tool tries to locate a possible solution by finding a mutant that gives the expected results for all the test cases defined in the file. This mutation is then logged as a possible valid

correction for the fault, while the engine continues to look for more possible solutions until all the mutation operators have been applied. The results are logged in .txt files containing useful information about which test cases managed to detect errors, which mutations were identified, the possible corrections for the initial code, etc.



```

1 test 3
2 2;4;5;1
3 5;6;3;1
4 2;1;2;2
5 3;1;2;4
6 4;8;9;1
7 3;1;7;1
8 4;4;4;3
9 5;5;5;3
10 5;5;3;2
11 6;7;2;1
12 10;2;1;4
13 10;3;8;1
14 10;5;-5;4
15 5;3;5;2
16 2;2;2;3
17 0;1;2;4
18 3;3;1;2
19 3;4;2;1

```

Figure 2. Format of the Test Cases File

C. Supported Mutation Operators

Mutations are performed at the method level using operators that are either arithmetic, relational or logical. At the class level, mutation is performed with operators applied to a class or a number of classes, and usually involves changing calls to methods or changing the access modifiers of the class characteristics (public, private, friendly etc.). The operators supported by the proposed mutation engine are the:

Arithmetic

- AOR_{BA} – arithmetic operations replacement (binary, assignment)
- AOR_S – arithmetic operations replacement (shortcut)
- AOI_S – arithmetic operations insertion (shortcut)
- AOI_U – arithmetic operations insertion (unary)
- AOI_A – arithmetic operations insertion (assignment)
- AOD_S – arithmetic operations deletion (shortcut)
- AOD_U – arithmetic operations deletion (unary)
- AOD_A – arithmetic operations deletion (assignment)

Relational

- ROR – relational operations replacement

Conditional

- COR – conditional operations replacement
- COI – conditional operations insertion
- COD – conditional operations deletion

Logical

- LOR – logical operations replacement
- LOI – logical operations insertion
- LOI_A – logical operations insertion (assignment)

- LOD – logical operations deletion
- LOD_A – logical operations deletion (assignment)

Shift

- SOR – shift operations replacement
- SOI_A – shift operations insertion (assignment)
- SOD_A – shift operations deletion (assignment)

Replacement

- PR – parameter replacement
- LVR – local variable replacement

For example if the AOR_{BA} operator is applied on the following line of code

```
jkreturn this.num / this.den;
```

then the result will be the creation of four different mutations by replacing the division (/) operator with either addition (+), multiplication (*), subtraction (-) and modulo (%). The four cases for the produced mutated line of code are shown below:

- (a) return this.num * this.den;
- (b) return this.num + this.den;
- (c) return this.num - this.den;
- (d) return this.num % this.den;

D. Specification-Based Mutations

The number of possible mutated programs for a certain case-study may be quite large depending on the type and number of statements in the source code. Mutations processing time is proportional to the number of mutants processed. This is a significant problem that may hinder the use of mutation testing in certain cases. There is a need to minimize mutation testing execution time. This is feasible if useless mutations are removed or avoided. Such mutations correspond to invalid forms of executions for that particular program which may be determined by the program's specifications. Therefore, the specifications must be taken into consideration when producing a mutant. These Specifications are implemented as CC in VS2010. This feature enhances the fault detection part of our tool, as it removes any possible mutations that do not satisfy the specifications defined in the source code.

```
public class Test {
    private int Foo(int a, int b) {
        Contract.Requires(a > b);
        Contract.Requires(b > 0);
        Contract.Ensures(Contract.Result<int>()>0;
        ...
        return (a / b);
    } ...
    private void Goo( ) {
        int x, y;
        ...
        x = y + 10;
        int result = Foo ( x , y) }
}
```

Figure 3. Class Test Example with CC specifications

Figure 3 demonstrates how mutations are driven by the specifications inserted via CC. Class *Test* includes methods *Foo* and *Goo* and uses CC to express two pre-conditions (denoted by *Contract.Requires*) and one post-condition (denoted by *Contract.Ensures*).

In *Goo* the assignment of *x* affects the values with which *Foo* is called. The first pre-condition requires that $x > y$. The engine normally would perform operation replacement substituting '+' with '-', '/', '%' and '*'. Due to the aforementioned pre-condition the engine will drop the first three replacements and use only the last one as it is the only replacement that will still satisfy the pre-condition. The same applies for $b > 0$, where any arithmetic replacement should not set *b* equal or less than zero. In this way the engine produces only valid mutations and ensures that a certain mutation is implemented in the engine which enables the production only of valid mutants thus ensuring that the minimum possible time and effort will be spent on the subsequent analysis and testing activities. This approach also limits the search for a possible solution by the user, when a number of solutions are identified by the engine.

E. The software tool

A dedicated software tool has been developed to support the process of MT. An example scenario is given below to demonstrate its operation: A source code file, the project file of the program tested and a test case file (optional) are given as input to the system. If no test case file is provided the program continues with only the creation of the mutations and nothing more. The project file and all the references to other files or libraries are automatically located and linked, and the source code file is compiled through the validation component. In the case of compilation errors a pop up window is presented to the user with the corresponding information (Figure 4) and the process is terminated.

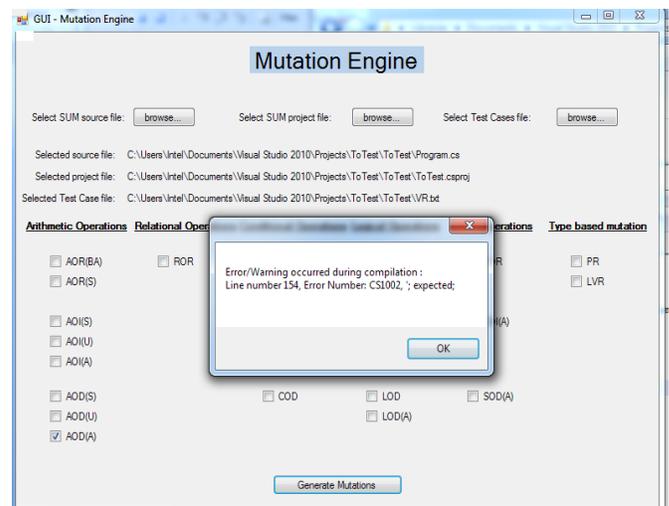


Figure 4. Execution : Errors in compilation

If there are just warnings, the user is again informed, but the system now continues to the next step. Static analysis of

the source code is performed, resulting in the creation of an AST. The visitor component then passes through the AST and creates the lists that store the information found in the source code (variables, classes, statements, etc.). The third component takes as input the lists created by the visitor and a set of mutators selected by the user, applies these operators and returns the resulting mutated programs in the path defined (Figure 5). The last component, the automatic test case quality assessment component, reads the test case file provided by the user and executes the initial program with those inputs. If the execution fails on any test case, the user is notified that the original program does not validate all the test cases correctly compared to their expected results and then searches for a mutated version that does. If the initial program validates correctly all test cases, then it continues with assessing the quality of the test data set in order to report the ability of the test data set to identify faults in the mutants.

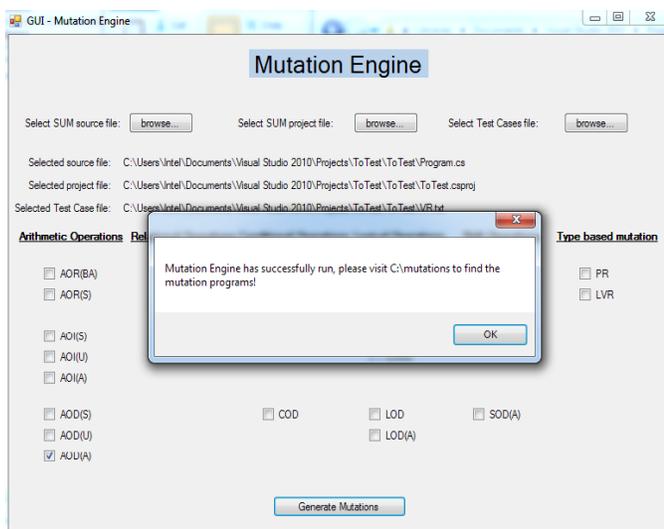


Figure 5. Execution : Mutations successfully produced

IV. EXPERIMENTAL RESULTS

In order to test the mutation engine and the corresponding trial a series of experiments were performed that would help us assess both the correctness and the efficiency of our testing approach using programs written in both the C# and VB programming languages. Four categories of experiment were conducted as follows:

Category A addressed the quality (adequacy) of a given test case set against a benchmark program. A file containing test cases and the expected results was fed into the tool along with the source code that verified all of the test cases. Then all the mutations produced by the engine were tested against the test cases, logging which mutants were discovered or which ones produced the same result as the original program.

Category B assessed the ability of the tool to discover a fault in the original code and provide a solution to correct it. In the case a test cases set fed into the tool does not validate the original program the tool continues to create mutations and propose possible solutions in order to validate all the test cases found in the test file, whilst satisfying all the specifications found in the code contracts.

Category C demonstrated that the proposed engine works as supposed on both C# and VB code, by producing correctly a number of mutations based on atomic changes to the source code according to the user's selected types of mutation operators. The same functions are developed in both programming languages and the mutations produced were compared. Also, both the results of C# and VB mutations were tested against a set of test cases to see if the mutation engine could identify the same mutants for both languages.

Category D evaluates the scalability of the proposed approach on large, real-world programs. Benchmark programs were used, and the type and number of mutations was recorded. It is worth mentioning that the experiments were performed on an Intel i7-2600 CPU at 3.4 GHz with 4 GB of RAM, while the programs used are available in various sites on the Internet (e.g., <http://www.c-program-example.com>).

Lastly, category E demonstrated that the mutation engine could eliminate the mutants that violate the pre-conditions, post-conditions or invariants set for a program. Comparisons of the number of mutations produced when using code that contains specifications against code that does not contain specifications.

The experiments are analysed below:

A. Test-Data Quality Assessment

This experiment used a specific benchmark program, the triangle classification program, which is shown in Figure 6. This program was tested against the 19 different test cases shown in Table I, the meaning of the values used in its last column is as follows: 1 equals to a scalene triangle, 2 equals to an isosceles triangle, 3 to an equilateral and 4 does not correspond to a triangle.

```
int triang(int i, int j, int k) {
    if ((i <= 0) || (j <= 0) || (k <= 0))
        return 4;
    int tri = 0;
    if (i==j)
        tri+=1;
    if (i==k)
        tri+=2;
    if (j==k)
        tri+=3;
    if (tri==0) {
        if ((i+j==k) || (j+k==i) || (i+k==j))
            tri=4;
        else tri=1;}
    else {
        if (tri>3) tri=3;
        else {
            if ((tri==1) && (i+j>k))
                tri=2;
            else{
                if ((tri==2) && (i+k>j))
                    tri=2;
                else {
                    if ((tri==3) && (j+k>i))
                        tri = 2;
                    else tri = 4; } } } }
    return tri; } }
```

Figure 6. Triangle Classification Program Source Code

Using the values in the first three columns of Table I for the corresponding variables it appears at first that the TCP has been adequately tested. The source code, the project file and the test cases file were fed into the tool and all of the available mutation operators were selected. After the engine finished both creating the mutants and testing them with the test case set, a general results file was created (Figures 7 and 8) which contained all the mutations and the verdict whether there is at least one test case in the test set that could identify the alteration performed or not. The file also includes the number of total mutations, the mutations that failed to compile, the number of mutations that were successfully discovered or not, the mutation score and the time needed for the engine to produce and test the mutations.

TABLE I. TEST DATA THAT COVER ALL POSSIBLE OUTPUTS OF THE TRIANGLE CLASSIFICATION PROGRAM (TCP)

i	j	k	Result
2	4	5	1
5	6	3	1
2	1	2	2
3	1	2	4
4	8	9	1
3	1	7	1
4	4	4	3
5	5	5	3
5	5	3	2
6	7	2	1
10	2	1	4
10	3	8	1

10	5	-5	4
5	3	5	2
2	2	2	3
0	1	2	4
3	3	1	2
3	4	2	1

Along with the general results file created at the end of the process, a results file for each mutation is created. This file contains the results of applying each of the test cases, to the program and which test case identified the error, if such a case exists.

The results show that 517 mutants were created, from which 58 could not be compiled so they were discarded. From the remaining of 459 mutants, 175 could not be identified by the test case set as they successfully yielded an identical result as the original program. Finally, 284 mutations were identified by at least one test case, leading to a mutation score of 61%.

This experiment demonstrates that the proposed engine is able to assess the quality of a set of data to adequately test a given program.

Such a change that yields the same result is the following change on the second statement of the code in Fig. 6:

```

if ((i <= 0) || (j <= 0) || (k <= 0))
    to
if ((i <= 0) || (j < 0) || (k <= 0))
    
```

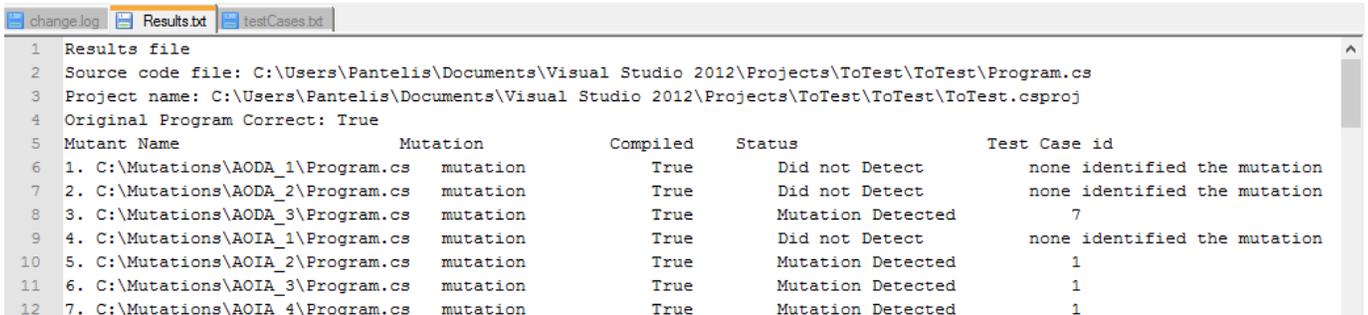


Figure 7. Beginning of general results file using Triangle Classification program written in the C# language

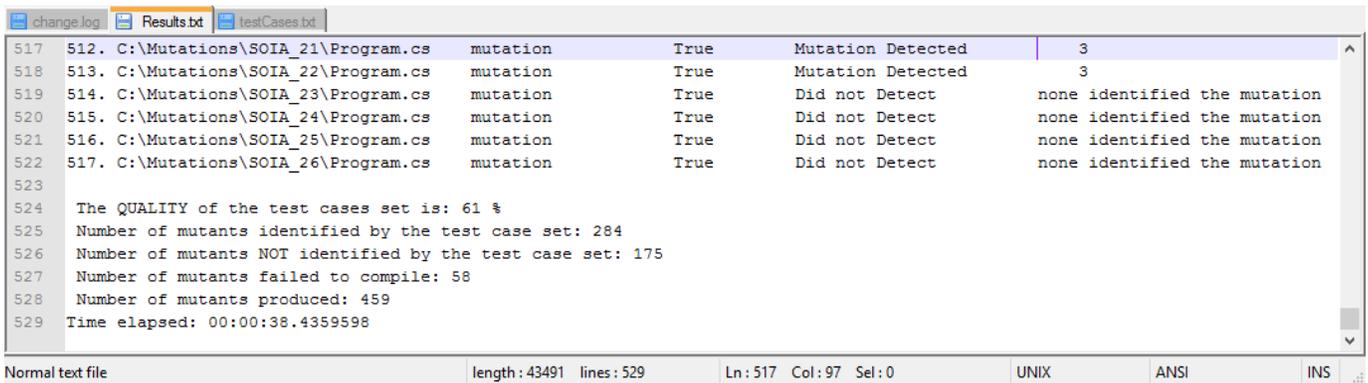


Figure 8. End of general results file using Triangle Classification program written in the C# language

B. Fault Detection

This set of experiments was concerned with the ability of the mutation engine to reveal errors that were injected in the initial source code of the triangle classification program. A number of faults were manually injected into the code and is described below.

Figure 9 shows two faults inserted in the code, one relational and one unary.

```
int triang(int i, int j, int k) {
    if ((i <= 0) || (j != 0) || (k <= 0)){
/**1. should have been ((i<=0) || (j<=0) || (k<= 0))**/
        return 4;}
    int tri = 0;
    if (i==j)
        tri+=1;
    if (i==k)
        tri+=2;
    if (j==k)
        tri+=3;
    if (tri==0) {
        if ((i+j==k) || (j+k<=i) || (i+k<=j))
            tri=4;
        else tri=1;}
    else {
        if (tri>3)
            tri+=3;
/** 2. should have been tri=3;**/
        else {
            if ((tri==1) && (i+j>k))
                tri=2;
            else{
                if ((tri==2) && (i+k>j))
                    tri=2;
                else {
                    if ((tri==3) && (j+k>i))
                        tri = 2;
                    else tri = 4; } } }
        return tri; }
}
```

Figure 9. Faults injected into Triangle Classification Program

The first change was the replacement of the \leq relational operand with \neq in the second line of the original code. The tool suggested 6 possible solutions (Table II). It's clear that correction #5 is the one that reverted the faulty program to the original version, but the other 5 proposed fixes yield the same results with number 5. This can be due to the quality of the test cases set and its inability to detect all the mutants in the first place. This means that it is possible to check only 6 out of the 615 working mutants to find a correct version. Consequently only 1% of the work is needed compared to checking all mutations for a possible correction.

The second alteration involved changing the assignment $tri=3$ to adding 3 to the variable ($tri+=3$). The engine applied all mutation operators in the original version and suggested 2 possible corrections for this case. The first was the arithmetic operations deletion (AOD_A_4) mutation, which concerned the deletion of the plus operand from the line changed. The second suggestion was the arithmetic operations

replacement (AORBA_33) that suggested the change of the + operand to $-$. This suggestion again resulted in a version that satisfied all test cases; as we can see from the code the assignment is executed when the value of tri is equal to 6, so subtracting 3 will result in tri taking the value of 3 and validating correctly the test cases.

TABLE II. PROPOSED FIXES FOR THE FIRST INJECTED FAULT

No.	Name	Proposed Fix
1	COI_2	if ((i <= 0) (!(j != 0)) (k <= 0))
2	COI_4	if ((i <= 0) !(j != 0) (k <= 0))
3	COR_1	if ((i <= 0) (j != 0) && (k <= 0))
4	COR_2	if ((i <= 0) && (j != 0) (k <= 0))
5	ROR_9	if ((i <= 0) (j <= 0) (k <= 0))
6	ROR_10	if ((i <= 0) (j == 0) (k <= 0))

The example of Figure 10 employs CC with three pre-conditions, one post-condition and one invariant, and involves two errors inserted in class `CompareParadigm` that cannot be traced by the static analyzer in VS2010.

```
class CompareParadigm {
    int num,den;

    public CompareParadigm(int numerator, int denominator){
        Contract.Requires(0 < denominator);
        Contract.Requires(0 <= numerator);
        Contract.Requires(numerator>denominator);
        this.num += numerator;
/** should have been this.num = numerator **/
        this.den = denominator;
    }
    [ContractInvariantMethod]
    private void ObjectInvariant() {
        Contract.Invariant(this.den > 0);
        Contract.Invariant(this.num >= 0);
    }

    public int ToInt() {
        Contract.Ensures(Contract.Result<int>()>=0);
        return this.num * this.den; }
    }
/** should have been this.num / this.den **/
}
```

Figure 10. CompareParadigm Class with embedded Code Contracts

The engine is once again capable of bringing these errors to light using the arithmetic operation replacement (AOR_{BA}) and arithmetic operations deletion (AOD_A) mutators.

C. C# and VB comparative evaluation and compatibility issues

The tool has been extended to support both C# and VB. In order to assess the behavior of both C# and VB versions of the triangle classification program were used and the results compared.

The choice of analyzing only the arithmetic mutation operators was made, as they produce a large number of mutants allowing the extraction of some safe conclusions.

TABLE III. MUTATED PROGRAMS CREATED BY THE ENGINE FOR VB AND C#

Mutation type	Number of Mutations	
	Visual Basic	C#
AOD _A	3	3
AOI _A	65	65
AOI _S	66	66
AOI _U	41	41
AORB _A	36	36
Total	211	211
Failed to Compile	22	0
Identified Mutations	70	125

Table III shows that the mutation engine produces the same number of mutations for each operator in both cases of coding languages. In C# all of the mutations were compiled successfully, but when dealing with the VB source file, 22 mutations out of 211 could not be compiled. Further investigation of the mutated VB code files that could not be compiled highlighted that the mutations produced a form of syntax that is not always allowed in VB. An example of such a case is the production of the line below:

```
tri %= 2
```

VB does not support the use of the % operator, as it uses the mod operator to divide two numbers and return their remainder. After carefully checking all 22 mutations that failed to compile the observation that all of them failed because of the use of the % operator was made.

Continuing this evaluation, a comparison between the results files of the C# and VB versions of the code revealed that all of the identified mutants of the VB version were included in the C# mutants as well. Focusing on cases that were identified in C# but not in VB all of them were cases where the ++ and -- operators were introduced before a variable, as for example:

```
if ( ++tri = 1 ),
```

or cases with the += and -= operators being introduced as in:

```
tri -= 2
```

The use of these four operators is something that VB's compiler does not report either as a warning or an error; therefore the corresponding statements are compiled correctly, but they have no meaning and functionality. Because these statements are ignored, the mutated program yields the same behavior as the original version.

All 55 mutations that were not identified by the engine were mutations that used the four operators. This is one of the main compatibility issues raised in the extension of the proposed engine and will be addressed in future work possibly by removing these operators when dealing with VB source code.

Further investigation of the implications of the VB support took place by assessing test cases for a sample program based on the Find Max function; the program takes as input four numbers and returns the largest one (Figure 11). This would

further validate the mutation engine's support for locating and correcting faults in VB programs.

```
Dim max As Integer = 0
If num1 > num2 Then
    max = num1
Else
    max = num2
End If
If max < num3 Then
    max = num3
    If max < num4 Then
        max = num3
    End If
Else
    If max < num4 Then
        max = num4
    End If
End If
```

Figure 11. FindMax Program implemented in VB Programming Language

The evaluation used a test case file that described 20 cases with their expected results. An excerpt of the file can be seen in Table IV.

TABLE IV. PART OF THE FINDMAX PROGRAM TEST CASES SET

Num1	Num2	Num3	Num4	Result
5	6	7	8	8
5	6	3	1	6
4	8	9	1	9
-9	-4	-2	-1	-1
3	4	2	1	4

The tool was executed and the selection of all of the available mutation operators to be applied on the source code was made. This resulted in the production of a total of 336 mutations, from which 84 failed to compile due to the reasons described previously. From the remaining 252 produced mutants, 130 were identified by at least one test case, while 122 were not, something that computes a mutation score of 51% (Table V), indicating that the engine was able to detect 51% of the produced mutations with the test cases set fed.

TABLE V. MUTATED PROGRAMS CREATED BY THE ENGINE FOR VB IMPLEMENTATION OF FINDMAX FUNCTION

Mutation type	Number of Mutations
	Visual Basic
AOI _A	65
AOI _S	28
AOI _U	14
COI	4
LOI	14
LVR	96
PR	69
ROR	20
SOI _A	26
Total	336
Failed to Compile	84
Identified Mutations	130

In order to complete the conclusions of the review of the VB support the fault locating part of the tool was further investigated. For doing so the next line of code was changed as shown below:

```
from   If max < num3 Then
to     If max > num3 Then
```

Again the same test cases set was fed into the engine, as before, and the results of the Mutation Engine proposed two Relational Operator Replacements that could possibly fix the fault. These were:

```
(i) If max < num3 Then
(ii) If max <= num3 Then
```

The first replacement brings the program to its original state (i.e., before injecting the fault), while the second one again yields the same results as the first, as it does not affect the rest of the code in a way that alters the results for any of the test cases. As seen in bold letters in Figure 11, in either of the two cases the result would be that variable `max` gets the contents of the `num3` variable.

In general, the extension of the engine with the VB support module, although presenting some compatibility issues to resolve in the future, provided some encouraging results showing that the proposed engine is quite useful for testing source code written in VB exhibiting comparable performance to that when using C# code.

D. Time Behavioral Analysis

The fourth category of experiments, involved time analysis and measurements on differently sized C# programs. To this end, replication of the code of the Triangle Classification program was decided, by 2, 4, 6 and 8 times producing double the size of the program in each case. The test case quality assessment module of the tool was used with the same test cases shown in Table I. The `Stopwatch` class of the `System.Diagnostics` library was used to measure the time needed to produce the mutations.

TABLE VI. BENCHMARKS ON C# CODE

Lines Of Code	Mutations	Time (seconds)
67	468	46
134	905	77
268	1677	157
402	2480	216
536	3290	299

Table VI indicates that the time and number of mutations increases almost linearly and proportionally to the number of lines of code. An apparent analogy exists between the three values: doubling the lines of code nearly doubles both the number of mutations and hence the time needed for the engine to create them, as well as to test them. Notably, approximately 50% of the mutants corresponded to the number of failed to detect mutations, which were executed (tried) at least 20 times each, while the rest of the “normal” mutations were run a

variable number of times, ranging from 1 to 20. This emphasizes the importance of controlling the number of “useless” mutants addressed by the proposed mutation engine via the specification-driven mutation production and evaluation, as explained in the next section. For example, in the first case shown on Table VI where 67 lines of code exist in the source file for which 468 mutations were produced, it took the mutation engine 46 seconds to generate and test the mutants. This is roughly 0.1 seconds spent on producing and testing each mutant. If a programmer would have to create manually the mutants and evaluate them against the test cases, he would have needed at least 2 minutes to make each change, compile the code and run it against all the test cases. Also, he would have to document the results and keep track of all the mutators applied, something which would have taken extra time as well. The benefits of the automatic tool against the manual creation and evaluation of the mutants are clear and significant in terms of the time and effort needed.

In summary, the proposed solution was successfully tested on a large number of automatically created errors injected in the code against a number of test cases, reporting the mutants that identified (or not) each error in a reasonable time span. The time was less than the time needed for manually creating modified versions of the initial code and testing them one by one using the test cases.

E. Normal vs Specifications-Based Mutations Production

The fifth category of experiments involved the use of the CC. Using CC the tool can eliminate the mutants that violate the pre-conditions, post-conditions or invariants set for a program.

First, class `CompareParadigm` listed earlier, which includes a number of code contracts, was selected for experimentation. The number of mutations produced with the use of the specifications was compared to that of the same class with no specifications defined in code (in this case the engine with the CC disabled). Table VII lists the number of mutations that were produced according to the mutation operator used. A 58% reduction in the number of mutants is achieved when using the code contracts version of the code, which resulted in the engine generating 16 mutants compared to 38 that were produced without taking into consideration the specifications.

TABLE VII. MUTATED PROGRAMS CREATED BY THE ENGINE WITH (SPECS-BASED) AND WITHOUT THE USE OF SPECIFICATIONS (NORMAL)

Operator	Number of Mutations	
	Specs-based	Normal
AOR _{BA}	5	8
AOI _S	7	10
AOI _U	0	6
LOI	2	6
PR	2	3
LVR	0	5
Total	16	38

This reduction is quite significant, as the code consisted of less than 20 statements. Therefore, one can safely argue that in cases of large programs the computational burden will be considerably eased, preserving the effectiveness and efficiency of the testing process. Moreover, when used in conjunction with the fault locating part of the engine, it will obtain a smaller number of solutions. Written specifications can be used to constrain the creation of mutant solutions and the tool can propose only one solution to fix the fault.

V. CONCLUSIONS AND FUTURE WORK

Software development is prone to producing lower than expected quality software while the chance of project failure is high. Software Testing is an important, though complex, area of software development that mainly affects the quality and reliability of delivered software systems. A high percentage of software development time is devoted to testing.

Automatic software testing approaches are increasingly popular among researchers. They develop effective methods for fault locating and debugging so as to reduce testing complexity and lead to faster and cheaper software development steps with high quality standards.

Mutation testing is a technique that produces different versions of a program under study, each of which differ slightly from the original one, often mimicking common mistakes that programmers tend to make. These mutated versions are used to either identify faults or to assess the adequacy of a given set of test cases. In this context, a this paper proposes a simple, yet efficient mutation engine, in which a user-selectable number of mutation operators can be applied at the method level and incorporating CC to generate only valid mutants based on the program's specifications. The engine is developed in the Visual Studio 2010 platform and utilizes Code Contracts to represent the specifications that must be satisfied with pre-conditions, post-conditions and invariants for both C# and VB programming languages.

The engine is supported by a dedicated software tool consisting of four main parts. The first part verifies the syntactical correctness of the source code and proper linking with the appropriate libraries. The second part statically analyses the source code using grammatical analysis and produces the Abstract Syntax Tree representation of the source code. The third part uses the information gathered from the AST and generates mutations using specific operators selected by the user and obeying the rules imposed by the encoded specifications. The last part is the test case assessment component which either calculates the quality of a given test cases set or proposes possible corrections of faults that exist in code.

Five series of experiments were conducted that showed that the mutation engine is a tool that may be used for identifying faults in the code and for assisting the creation of the proper set of test data, both in C# and VB. Furthermore, the experiments demonstrated that the engine scales up smoothly as programs become larger in a time effective manner for creating and testing the mutants. Lastly, the incorporation of specification-based concepts allows for the significantly improved performance of the mutation engine by

reducing the number of mutants processed and solutions proposed according to the desired functionality expressed in the specifications, thus saving time and effort.

Future work will involve extending the proposed mutation engine to include more class-level mutators. Further additions and enhancements will be performed for both the C# and VB modules of the tool, while for the VB support the problem of applying mutators that produce invalid statements will be addressed. Moreover, integration of the engine with tools offered by the VS2010 is under investigation such as PEX, which is responsible for unit testing in order to automatically create test cases sets that have high code coverage [20] and UModel, which assists in creating UML diagrams. The UML diagrams from UModel can then generate source code that incorporates specifications that were set in the diagrams. This integration will enable the formation of a complete testing environment with dynamic user interaction, both at the flow of control level and at the diagrammatical level.

Our work can be compared only to a limited number of similar studies in literature: Saleh and Kulczycki [21] investigated how formal specifications can detect implementation errors in C# with the use of Creator of Mutants (CREAM) tool [22] and Boogie verifier [23] of SPEC# specifications. Their work tries and succeeds in showing how formal methods can affect the creation of bug-free programs by assessing their ability to detect design-time errors based on the SPEC# specifications. They concentrate on identifying faults created by mutations, which do not satisfy the specifications, at design level, without the need for executing the code. Our approach has a different purpose as it aims at assessing the quality of a test case set to identify faults in code and propose corrections for them. CCs are used, instead of SPEC#, for defining specifications which are verified dynamically upon code execution against a test case set and reports on any input values that do not satisfy the specifications. This makes possible the elimination of mutants that, although their code verifies statically the specifications, their execution against specific input values fails those specifications. Also our choice of CCs over SPEC# provides support for specifications in any language offered by VS, while SPEC# is designed to work only with C# code.

For MT in Java the work of Nica et al. [24] tries to answer the question if MT is really suitable for use in real-world environments. They evaluate the use of three different mutation tools for Java, MuJava, Jumble and Javalanche on some of the Eclipse IDE's source code, while they use the included JUnit tests provided with the source code on the Eclipse's repository to evaluate them. They neither try to locate and fix faults in code, nor do they assess the quality of the test cases set. Also, they use Java source code, while our work proposes a mutation engine to be used with both C# and VB programming languages.

Further validation of the proposed mutation engine will take place with the use of projects developed by graduate students. This will enable a more systematic evaluation of the engine using programs of different size and complexity that will include real faults made by programmers, while assessing various parameters, such as the time for creating and processing mutations, the type of mutators used and the nature

of the errors introduced. This systematic investigation will also bring to light any scalability issues not detected in this version of the engine. Moreover, efforts for increasing the performance of the Mutation Engine will be made with the use of parallel programming and multithreading, coupled with benchmarking tasks on a variety of different processing power systems. Lastly, the problem of regression faults will be addressed by exploring the feasibility of providing a correction to more than one fault without affecting any previous corrections.

REFERENCES

- [1] A.S. Andreou and P. Yiasemis, "A Specifications-Based Mutation Engine for Testing Programs in C#", Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA), Barcelona, Spain, 2011, pp. 70-75.
- [2] C. Kaner, J.H. Falk and H.Q. Nguyen, "Testing Computer Software", John Wiley & Sons Inc., New York, NY, USA, 1999.
- [3] A. Bertolino, "Software testing research: achievements, challenges, dreams", Proceedings of 29th International Conference on Software Engineering (ICSE 2007): Future of Software Engineering (FOSE'07), Minneapolis, USA, 2007, pp. 85-103.
- [4] B. Gauf and E. Dustin, "The case for Automated Software Testing", Future Directions in Software Engineering Journal, Vol. 10 (3), 2007, pp. 29-34.
- [5] R. Patton, "Software Testing", Sams Publishing, 2nd edition, 2006.
- [6] M.E. Khan, "Different Forms of Software Testing Techniques for Finding Errors," International Journal of Computer Science Issues (IJCSI), 2010.
- [7] "Mutation Testing Repository", <http://www.dcs.kcl.ac.uk/pg/jiayue/repository/>, [accessed 10 May 2011].
- [8] M. Nica, S. Nica and F. Wotawa, "On the use of mutations and testing for debugging," Software-Practice and Experience, Article published online, 2012.
- [9] R.A. DeMillo, R.J. Lipton and F.G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer", IEEE Computer Vol. 11(4), 1978, pp. 34-41.
- [10] R.G. Hamlet, "Testing Programs with the Aid of a Compiler", IEEE Transactions on Software Engineering, Vol. 3(4), 1997, pp. 279-290.
- [11] M. Harman, Y. Jia and W.B. Langdon, "Strong Higher Order Mutation-Base Test Data Generation", ESEC/FSE'11, Szeged, Hungary, September 5-9, 2011.
- [12] A.J. Offutt, "The Coupling Effect: Fact or Fiction", ACM SIGSOFT '89 - Third symposium on Software testing, analysis, and verification ACM, New York, USA, 1989.
- [13] G. Fraser and A. Zeller, "Generating Parameterized Unit Tests", International Symposium on Software Testing and Analysis (ISSTA'11), Toronto, Canada, July 17-21, 2011.
- [14] A.A. Sofokleous and A.S. Andreou, "Automatic, Evolutionary Test Data Generation for Dynamic Software Testing", Journal of Systems and Software, Vol. 81(11), 2008, pp. 1883-1898.
- [15] C.C. Michael, G. McGraw and M.A. Schatz, "Generating software test data by evolution", IEEE Transactions on Software Engineering (12), 2001, pp. 1085-1110.
- [16] "Visual Studio 2010", (2009) <http://www.microsoft.com/visualstudio/en-us/products/2010-editions>, [accessed 18 May 2011].
- [17] "Code Contracts User Manual", (2010), Microsoft Corporation, <http://research.microsoft.com/en-us/projects/contracts/userdoc.pdf> [accessed 20 May 2011].
- [18] "SPEC#", (2004), <http://research.microsoft.com/en-us/projects/specsharp/>, [accessed 04 August 2012].
- [19] "SharpCode - The Open Source Development Environment for .NET", (2009), <http://www.icsharpcode.net/opensource/sd/>, [accessed 17 May 2011].
- [20] "Pex and Moles - Isolation and White box Unit Testing for .NET", (2004), <http://research.microsoft.com/en-us/projects/pex/>, [accessed 04 August 2012].
- [21] I. Saleh and G. Kulczycki, "Design-Time Detection of Implementation Errors Using Formal Code Specification", RESOLVE 2010 Workshop: Advances in Automated Verification, Denison University, Granville, Ohio, June 8, 2010.
- [22] A. Derezinska and A. Szustek, "CREAM - a System for Object-oriented Mutation of C# Programs", Information Technologies, Vol.13 (5), Gdansk, 2007, pp. 389-406.
- [23] "Boogie: An Intermediate Verification Language", <http://research.microsoft.com/en-us/projects/boogie/>, [accessed 08 December 2012].
- [24] S. Nica, R. Ramler and F. Wotawa, "Is Mutation Testing Scalable for Real-World Software Projects?", Third International Conference On Advances in System Testing and Validation Lifecycle (VALID), Barcelona, Spain, 2011.

Benchmarking Data as a basis for Choosing a Business Software Systems Development and Enhancement Project Variant – Case Study

Beata Czarnacka-Chrobot

Department of Business Informatics

Warsaw School of Economics

Warsaw, Poland

e-mail: bczarn@sgh.waw.pl

Abstract—Execution of Business Software Systems (BSS) Development and Enhancement Projects (D&EP) is characterised by the exceptionally low effectiveness, leading to the considerable financial losses. Thus, it is necessary to rationalize investment decisions made with regard to the projects of this type. Each rational investment decision should meet two measurable criteria: effectiveness and economic efficiency. In order to make *ex ante* evaluation of these criteria, being key to the decision-making process, one may successfully use ever richer resources of benchmarking data, having been collected in special repositories that were created with improvement of software processes and products in mind. The goal of this paper is to present possibilities of employing benchmarking data in the rationalization of investment decision concerning the choice of BSS D&EP execution variant on the basis of a case study. Thanks to the rational investment decisions made on the basis of reliable and objective benchmarking data it is possible to reduce losses caused by the low effectiveness of BSS D&EP. These issues classify into economics problems of software engineering.

Keywords—software engineering economics; business software systems development and enhancement projects variants; rational investment decision; effectiveness; efficiency; benchmarking data; case study

I. INTRODUCTION

In practice, execution of Business Software Systems (BSS) Development and Enhancement Projects (D&EP) is characterised by the exceptionally low effectiveness, leading to the considerable financial losses (the paper is an extended version of [1]). This may be proved by numerous analyses. As indicated by the results of the Standish Group studies success rate for application software D&EP has never gone beyond 37% [2], while products delivered as a result of nearly 45% of them lack on average 32% of the required functions and features, the estimated project budget is exceeded by approx. 55% on average and the planned project time – by nearly 80% on average [3] (for more details see [4]). Analyses by T.C. Jones plainly indicate that those software D&EP, which are aimed at delivery of business software systems, have the lowest chance to succeed [5]. The Panorama Consulting Group, when investigating in their 2008 study the effectiveness of ERP (Enterprise Resource Planning) systems projects being accomplished worldwide revealed that 93% of them were

completed after the scheduled time while as many as 68% among them were considerably delayed comparing to the expected completion time [6]. Merely 7% of the surveyed ERP projects were accomplished as planned. Comparison of actual versus planned expenses has revealed that as many as 65% of such projects overran the planned budget. Only 13% of the respondents expressed high satisfaction with the functionality implemented in final product while in merely every fifth company at least 50% of the expected benefits from its implementation were said to be achieved. Three years later, the respondents of Panorama Consulting Group study indicated that there were significantly more companies with ERP project overruns in 2010 than in 2009 [7].

Similar data, proving unsatisfactory effectiveness of BSS D&EP, are brought by the studies carried out in 2011 among providers of such projects in Poland [8]. According to the results, 80% of the surveyed organizations admit that the projects exceed the planned budget, 79% - that they exceed the planned execution time while 64% - that the quality assumptions for software products are not being met. In this case it results from the fact that slight percentage of providers manages the software systems development processes properly. What is interesting, all those numbers increase if the so-called expert methods are used to estimate project attributes – instead of estimates being based on standards and benchmarking data (most preferably own ones).

Meanwhile BSS are not only one of the fundamental IT application areas; also their development/enhancement often constitutes serious investment undertaking: spending on BSS may considerably exceed the expense of building even 50-storey skyscraper, roofed football stadium, or cruising ship with a displacement of 70.000 tons [9]. Yet quite often client spends these sums without supporting his decision on getting engaged in such investment by proper analysis of the costs, based on the rational, sufficiently objective and reliable basis. What is more, in practice COTS (Commercial-Off-The-Shelf) BSS rarely happen to be fully tailored to the particular client business requirements therefore their customization appears vital (see also [4]).

Exceptionally low effectiveness of BSS D&EP as compared to other types of IT projects (i.e., maintenance, support, package acquisition, implementation projects, projects delivering other types of software), especially with

their costs being considered, leads to the substantial financial losses, on a worldwide scale estimated to be hundreds of billions of dollars yearly, sometimes making even more than half the funds being invested in such projects. The Standish Group estimates that these losses – excluding losses caused by business opportunities lost by clients, providers losing credibility or legal repercussions – range, depending on the year considered, from approx. 20% to even 55% of the costs assigned for the execution of the analysed projects types (see e.g., [10], [11]). If direct losses caused by abandoning the BSS D&EP result from erroneous allocation of financial means, usually being not retrievable, in the case of overrunning the estimated cost and/or time, however, they may result from delay in gaining the planned return on investment as well as from decreasing it (necessity to invest additional funds and/or cutting on profits due to the overrunning of execution time and/or delivery of product incompatible with requirements) (for more details see [12]). On the other hand, analyses of The Economist Intelligence Unit, which studied the consequences of BSS D&EP delay indicate that there is strong correlation between delays in delivery of software products and services and decrease in profitability of a company therefore failures of BSS D&EP, resulting in delays in making new product and services available and in decreasing the expected income, represent threat also to the company's business activity [13].

What is more, the Standish Group studies also indicate that "the costs of these (...) overruns are just the tip of the proverbial iceberg. The lost opportunity costs are not measurable, but could easily be in the trillions of dollars. [For instance - B.C.C.] the failure to produce reliable software to handle baggage at the new Denver airport is costing the city \$1.1 million per day" [14]. These losses result from the insufficient level of the delivered product compatibility with the client's requirements as to the functions and features: over 1994-2010 an average conformity of this type never went beyond 70%, which means that the delivered applications lacked at least 30% of the specified functions and features [3]. Incompatibility of the delivered product with the required one proves to be the highest for large projects, in case of which the delivered product lacks on average even 60% of the required functions and features. While for medium- and small-sized projects such incompatibility amounts to approx. 35% and approx. 25% of functions and features, respectively.

The above studies unequivocally indicate there is a significant need to rationalize investment decisions made with regard to BSS D&EP. To do so, one may successfully use ever richer resources of benchmarking data, having been collected with the intention to support improvement of various IT projects, including BSS D&EP, in special repositories (for more details see [15]). The goal of this paper is to present possibilities of BSS D&EP investment decision rationalization with the use of benchmarking data, illustrated with an example taken from development practice. This decision concerns choosing variant of BSS D&EP execution – since each project of this type may be executed using one of the three variants, namely: (1) developing new BSS from scratch, (2) customization of

COTS BSS, and (3) modernization of BSS being currently used.

The paper is structured as follows: in Section 2 the author presents the criteria of rational investment decision in the context of BSS D&EP along with the selected results of studies concerning *ex ante* evaluation of these criteria. Section 3 is devoted to the presentation of the considered case study problem. In Section 4 the main conclusions coming from the benchmarking data analysis are pointed out, while in Section 5 the effectiveness and efficiency factors for the recommended BSS D&EP variant are analyzed. Finally, in Section 6 the author draws conclusions and some open lines about future work on the usefulness of benchmarking data, not only in the context of rationalization of BSS D&EP investment decision.

II. RATIONAL INVESTMENT DECISION CRITERIA FOR BUSINESS SOFTWARE SYSTEMS DEVELOPMENT AND ENHANCEMENT PROJECTS

Each rational investment decision should meet three criteria, which in the context of BSS D&EP should be interpreted as follows:

- Criterion of consistency, which means that the project undertaken should comply with the environment (economic, organizational, legal and cultural) – unlike the other two criteria, this criterion is not subject to quantitative assessment therefore it is skipped in this paper.
- Criterion of economic efficiency, meaning that the decision should benefit to the maximisation of the relationship between the effects to be gained as a result of project execution and the costs being estimated for the project.
- Criterion of effectiveness, meaning that such decision should contribute to achieving the assumed result, in the case of BSS D&EP usually being considered as delivering product meeting client's requirements with regard to functions and features without budget and time overruns.

Generally speaking, in the case of economic efficiency evaluation, effects are compared against costs necessary to achieve these effects while in the case of effectiveness evaluation these are only the results that are of significance. Thus, economic efficiency is measured by relating total effects to total costs. Meanwhile, effectiveness is measured by the ratio of the achieved result to the assumed result, which is being conveniently expressed as a percentage.

Both economic efficiency criterion as well as effectiveness criterion are based on the obvious assumption that the effects, costs and results are measurable. However, in the case of BSS D&EP this assumption is often treated as controversial. Numerous studies indicate that evaluation of BSS D&EP economic efficiency is made relatively rarely while fundamental reason for this *status quo* are difficulties related to identification, and most of all quantitative expression, of benefits resulting from the execution of such projects (see e.g., [16], [17], [18], [19], [20]). These studies reveal that difficulties related to identification and

quantitative expression of BSS D&EP costs too are of significance, which also is of importance to the evaluation of their effectiveness.

Key conclusions coming from the above mentioned studies have also been confirmed by the results of studies carried out by the author of this paper in two research cycles among Polish dedicated BSS providers (for more details see [21]). They revealed that at the turn of the years 2005/2006 the results obtained with the use of the effort estimation methods, employed only by approx. 45% of the respondents, were designed for estimating BSS D&EP costs and time frame while relatively rarely they were used to estimate economic efficiency – such use of these methods was indicated by only 25% of those using effort estimation methods. Heads of IT departments in Polish companies, for which BSS D&EP are executed, still explain the sporadically required calculation of this type of investments efficiency mostly by the necessity to undertake them – most often due to the fact that without such solutions they lack possibility to match competition from foreign companies, as well as to match foreign business partners requirements. While Polish public administration institutions in practice still do not see the need for the BSS D&EP economic efficiency evaluation, in most cases as an argument giving the non-economic purposes of systems being implemented in this type of organizations. On the other hand, at the turn of the years 2008/2009 the results obtained with the use of the BSS D&EP effort estimation methods (approx. 53% of BSS providers surveyed in this cycle declared they commonly employed such methods) were more often used to estimate efficiency: there was an increase to approx. 36% of those using effort estimation methods. This applies to internal IT departments of Polish companies yet still it does not comprise public administration institutions. This increase may be explained first of all by stronger care about financial means in the times of recession, however it still leaves a lot to be desired. Meanwhile, to rationalize various BSS D&EP investment decisions, one may successfully use benchmarking data, having been collected in special repositories with intention to support effective and efficient execution of such projects.

III. CASE STUDY: DESCRIPTION OF THE PROBLEM

A company that was facing the need to choose an appropriate variant of BSS D&EP execution collects and processes, as a part of its basic activity, orders for certain goods from all over the world in a 24-hour mode, 7 days a week through: website, client service centres, fax and electronic mail (description of the case study taken from [22]). All those channels cooperate with the application, having been functioning in the company for a dozen or so years already, designed for orders processing and which is no longer able to satisfy present requirements since:

- Large part of processes is not automated, which requires additional work for registering orders and that generates losses.
- Current status of orders is not known therefore they are being lost; as a result of this other losses are also

borne, which together with earlier mentioned losses are estimated to be approx. USD 5000 a day.

- System is expensive and difficult to maintain, with frequent malfunctions as it employs obsolete technology.
- System extends the time of delivering new products to the market, increases the risk of losing clients and lack of compliance with their requirements, slows down the growth of competitive advantage.

Thus, the company has faced a decision on choosing variant of BSS D&EP execution that would:

- Eliminate the above mentioned drawbacks of the existing solution.
- Contribute to short- and long-term profits – that’s why the costs and duration of project are of great significance.
- Reduce the costs of functioning of both company and technology.
- Contribute to the reduction of risk, both in terms of business and technology.

TABLE I. PARAMETERS OF OFFERS CONCERNING EXECUTION OF PARTICULAR VARIANTS OF BSS D&EP CONSIDERED

Variant	BSS D&EP variant	Execution cost offered	Execution time offered
1	Development of new BSS from scratch using modern technologies	USD 10 million	3 years
2	Customization of BSS purchased	USD 5 million	2 years
3	Modernization of BSS used currently	USD 3,5 million	1,5 years

Source: Author’s analysis based on [22, p. 2].

Offers for each BSS D&EP variant were submitted, having approximate average values as shown in Table I.

Since each variant was backed by certain part of the board and key users, an analysis aimed at supporting decision-making process was carried out.

IV. CONCLUSIONS FROM THE BENCHMARKING DATA ANALYSIS

The analysis used benchmarking data for BSS D&EP having been collected in the following repositories:

- Standish Group, featuring data about over 70 thousands of the accomplished application software D&EP, which were analysed using the tool called *VirtualADVISOR* [22].
- Software Productivity Research (SPR), containing data from approx. 15 thousands of the accomplished application software D&EP, which were used to verify conclusions coming from Standish Group repository analysis with the use of *SPR Knowledge Plan* tool [23].

- International Software Benchmarking Standards Group (ISBSG), having collected data from approx. 5 thousands of the accomplished application software D&EP [24], also used to verify findings coming from Standish Group repository analysis and also with the use of *SPR Knowledge Plan* tool, which at its present version offers possibility to import data from the ISBSG repository.

Priority was given to the Standish Group data and this being not only due to the size of this repository, objectivity of data (they come solely from clients) or the fact of IT branch appreciating its practical value [10] but also because they take into account an appropriate kind of client (in terms of branch and size of a company), appropriate kinds and size of BSS D&EP as well as appropriate type and size of application. Thus, using the Standish Group repository made it possible to match all three kinds of BSS D&EP against the profile, with 90% match of the 120 attributes of more than 100 projects [22].

What is also important, in their analyses the Standish Group employ clearly defined criteria of project classification, dividing projects into the following three groups (see e.g., [3], [11], [25]):

- Successful projects – that is projects completed with delivery of product having functions and features being in accordance with client requirements specification and within the estimated time and budget.
- Challenged projects – that is projects completed with delivery of product that is operating yet has fewer vital functions/features comparing to the client requirements specification and/or with overrun of the planned budget and/or duration.
- Failed projects – that is projects that were abandoned (cancelled) at some point of their life cycle or were completed with delivery of product that had never been used.

The Standish Group conducts its researches mostly from the point of view of the so-called success coefficient, which describes the share of successful projects in the total number of analysed projects completed during given year. What represents counterbalance to the projects comprised by the success coefficient are projects that ended with total or partial failure, i.e., failed and challenged projects. In case of challenged projects, this is also degree of fitness of the delivered product to the functions and features required by a client. Since the mid-1990s these numbers have shaped as shown in Table II.

In the analysis of the Standish Group data, the following criteria were employed as equivalent for particular variants of the BSS D&EP considered:

- 1) Criterion of expected BSS D&EP effectiveness, including:
 - a) chance to succeed
 - b) level of planned costs overrun
 - c) level of planned duration overrun.

- 2) Criterion of expected BSS D&EP efficiency, including:
 - a) return on investment (ROI)
 - b) payback period.

TABLE II. AVERAGE EFFECTIVENESS OF APPLICATION SOFTWARE D&EP EXECUTION OVER 1994-2010

Data for	Success coefficient (in %)	Partial failure (in %)	Total failure (in %)	Partial and total failure (in %)
1994	16	53	31	84
1996	27	33	40	73
1998	26	46	28	74
2000	28	49	23	72
2002	34	51	15	66
2004	29	53	18	71
2006	35	46	19	65
2008	32	44	24	68
2010	37	42	21	63

Source: [2] and [3].

Data presented in Table III clearly indicate that in the case being considered the highest chance to succeed is held by modernization variant, for which success coefficient is several times higher than that characteristic of variant consisting in development of new application, being only 4% (sic!), and significantly higher than that of COTS customization variant. Also in case of variant 3 the lowest percentage of projects ends with being abandoned – it is several times lower than in case of variant 1 and two times lower than in case of variant 2. What seems interesting, the highest percentage of projects that ended in partial failure (challenged projects) occurs in case of the customization of COTS application. What is more, the average expected overrun of both costs (see Table IV) and project duration (see Table V) is also the highest in case of this project variant.

TABLE III. EXPECTED CHANCE TO SUCCEED FOR PARTICULAR VARIANTS OF BSS D&EP CONSIDERED

Resolution	Variant 1	Variant 2	Variant 3
Successful	4%	30%	53%
Challenged	47%	54%	39%
Failed	49%	16%	8%

Source: [22, p. 4].

Moreover, data in Table IV clearly indicate that the average expected overrun of the planned costs for projects that ended in partial failure too is the lowest in case of variant 3. Also the lowest percentage of such projects overruns the costs by more than 50%. If offered costs and average expected overrun of these costs are taken into consideration when calculating the expected cost then it appears evident that the lowest expected cost of project execution applies to modernization variant.

TABLE IV. EXPECTED LEVEL OF PLANNED COST OVERRUN FOR PARTICULAR VARIANTS OF BSS D&EP CONSIDERED (CHALLENGED PROJECTS)

Cost overrun	Variant 1	Variant 2	Variant 3
0% to 50%	64%	58%	75%
51% to 50%	36%	42%	25%
Average	44%	47%	34%
Offered cost	USD 10 million	USD 5 million	USD 3,5 million
Estimated cost	USD 14,4 million	USD 7,35 million	USD 4,7 million

Source: Author's analysis based on [22, p. 4].

Analogous conclusions may be drawn on the basis of the analysis of data presented in Table V. Again, the average expected overrun of the planned duration for projects that ended in partial failure proves being the lowest for variant 3. Also the lowest percentage of such projects overruns the duration by more than 50%. If we take into account the offered duration and average expected overrun of this duration then we can see that the lowest expected duration of project execution applies to modernization variant too.

TABLE V. EXPECTED LEVEL OF PLANNED DURATION OVERRUN FOR PARTICULAR VARIANTS OF BSS D&EP CONSIDERED (CHALLENGED PROJECTS)

Duration overrun	Variant 1	Variant 2	Variant 3
0% to 50%	57%	59%	80%
51% to 50%	43%	41%	20%
Average	44%	45%	29%
Offered duration	36 months	24 months	18 months
Estimated duration	52 months	35 months	23,5 months

Source: Author's analysis based on [22, p. 4].

Data shown in Table VI clearly indicate that the highest percentage of projects characterised by the highest ROI can be found in case of variant 3 again. On the other hand, what is interesting is that projects with average ROI most often are projects consisting in developing new application from scratch while the lowest percentage of projects characterised by the lowest ROI can be found in case of customization variant.

TABLE VI. EXPECTED ROI FOR PARTICULAR VARIANTS OF BSS D&EP CONSIDERED

ROI	Variant 1	Variant 2	Variant 3
High	11%	34%	52%
Average	66%	57%	37%
Low	23%	9%	11%

Source: [22, p. 5].

In Table VII both ROI and payback period for particular variants of the considered project were estimated in

optimistic and pessimistic version. In the optimistic version it was assumed that the costs were identical with the offered costs while in the pessimistic version - that the costs were exceeded by the average values being expected for each variant analysed (see Table IV). Based on these assumptions, both in optimistic and in pessimistic version, the highest 5-year gain applies to the modernization variant; also in case of that variant the payback period proves the shortest. It is worth noting that project in variant consisting in developing the new application would pay off after nearly 5 and half years in the optimistic version and after nearly 7 and half years in the pessimistic version.

TABLE VII. EXPECTED ROI AND PAYBACK PERIOD FOR PARTICULAR VARIANTS OF BSS D&EP CONSIDERED

Variant	Optimistic version			Pessimistic version		
	Costs (in \$ millions)	5-year gain (in \$ millions)	Payback period (in years)	Costs (in \$ millions)	5-year gain (in \$ millions)	Payback period (in years)
1	10	0	5,4	14,4	0	7,3
2	5	7,25	3,2	7,35	2,8	4,4
3	3,5	10,6	2,4	4,69	7,9	3,1

Source: Author's analysis based on [22, p. 5].

The above analysis clearly indicates that what in the considered case would be the best of the three BSS D&EP variants both from the perspective of the expected effectiveness and from the perspective of the expected efficiency is variant consisting in modernization of the application being used (variant 3).

V. THE EFFECTIVENESS AND EFFICIENCY FACTORS FOR THE RECOMMENDED VARIANT

In the analysed case, BSS D&EP consisting in modernization of application being used proves the most effective as well as the most efficient, what results, among others, from (see also [22]):

- Undertaking of such projects as a rule is a result of clearly defined needs of users therefore their goals are comprehensible, what undoubtedly promotes users' engagement in the project and the board's support for the project, which, according to the list of success factors having been developed by the Standish Group since 1995, are still the two most important success factors [25].
- The fact that modernization projects do not require extensive analysis of requirements, numerous agreements, long-time training, changes of processes that would be destabilizing the work.
- Commonness of such projects thus, the skills of executing them are high; what is more, projects of this type do not require additional skills in terms of project management, they rather require technical, the so called „hard", skills.
- Present structure of project costs in terms of development activities, which due to the increased complexity of projects and ever more developed tools has changed and is now in inverse proportion

to the structure as it was 25 years ago: now programming costs make up approx. 20% while other development works make up approx. 80% of the total cost.

- The fact that modernization projects are characterised by the lowest hidden cost (mainly user’s time), estimated to be 15% of project costs versus 55% for variant 2 and versus 35% for variant 1.
- The discussed projects do not have redundant requirements – as this is the case of the COTS customization where, according to the Standish Group data, less than 5% (sic!) of the features and functions get used [22], and of the development of new products (see Figure 1).

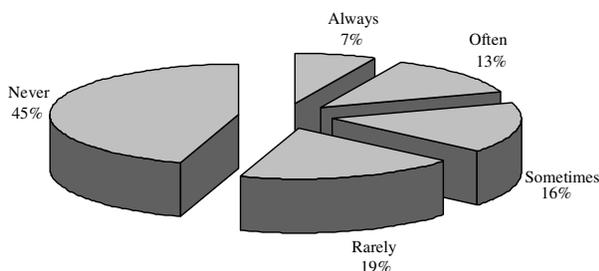


Figure 1. Average use of functions and features in the implemented software systems - custom development applications
Source: Author’s analysis based on [22, p. 15].

- Products smaller than those in case of developing application from scratch are developed as a result of the modernization projects and this is what increases their chance to succeed. Smaller products are usually delivered as a result of smaller projects.
- The discussed projects may be successfully carried out using agile approach, which also ranks high in the current list of Standish Group success factors [25]. The main objective of agile models is to quickly develop software that would be working correctly and this being thanks to focusing on strictly construction activities and keeping other activities down to a minimum, and not methodologically correct execution of that process instead [26]. Agile models were mostly developed for small and medium projects and they are used in rather small teams wherein there is no communication problem. They also require diverse and extensive knowledge and experience of the team members, and stable teams, located and working in one place throughout the project. Permanent accessibility of client’s representatives is also necessary. The Software Productivity Research study found that in the area of construction models usage there are some trends associated with industries and forms of software, e.g., business software systems and web applications

are more likely to use agile models than are systems software projects and military projects [27, pp. 6-7].

As far as the two last mentioned factors are concerned, it should be stressed that according to the analyses of Standish Group [11, p. 4], Software Productivity Research [28], and ISBSG [29, p. 2] this is minimisation of project size, first of all caused by the agile models having been used more and more often over time, that is responsible for the improvement of the application software D&EP effectiveness over 1994-2010: during that time the success coefficient has increased from 16% to 37% as both partial and total failure have decreased (see Table II). Failure to deliver functions and features required by a client also decreased (from 40% to 32%), as did the planned time overrun (from 164% to 79% - more than doubly) as well as planned budget overrun (from 180% to 55% - over three times) ([2], [14]). Thus, the losses resulting from the low scale of application D&EP effectiveness have shrunk considerably: from approx. USD 140 million to approx. USD 55 million, which accounts for the decrease of loss level from approx. 55% of the means invested in considered projects to approx. 20-25% of such means. Conclusion on the positive influence of the project sizes minimisation on their effectiveness gets confirmed by the data shown in Tables VIII and IX.

TABLE VIII. THE EFFECTIVENESS OF APPLICATION SOFTWARE D&EP EXECUTION BY PROJECT SIZE

Project size – measured by the work cost (in USD millions)	Success coefficient (in %)	Partial failure (in %)	Total failure (in %)
Over 10	0	11	19
6 – 10	6	20	28
3 – 6	13	36	39
0,75 – 3	19	18	8
Under 0,75	62	15	5
Total	100	100	100

Source: [25, p. 21].

As it may be seen in Table VIII, over 80% of the successful projects demonstrate having work cost up to USD 3 million whereas the work cost in nearly 20% of such projects ranges from USD 3 to 10 million. On the other hand, data in Table IX indicate that for application D&EP costing below USD 750 000, the chance to succeed is as high as 71% while for projects having work cost over USD 10 million it is barely 2%.

Meanwhile, comparison of the effectiveness of projects execution using agile models versus that using waterfall model (see Table X) leads to the conclusion that in case of agile projects the success coefficient is significantly higher yet still far from being regarded as satisfactory – even if we take into account that in this case it is above the average success coefficient. Partial failure too is lower in case of agile approach, however it almost equals its average value. Also, what definitely is worth pointing out is the fact that total failure both in case of agile and waterfall model is significantly lower than the average: for agile models – twice. In case of waterfall model smaller scale of total failure

does not have influence on increasing success coefficient but it does on increasing partial failure instead (such kind of failure, however, generates smaller losses than total failure does).

TABLE IX. THE APPLICATION SOFTWARE D&EP SIZE BY THE PROJECT EXECUTION EFFECTIVENESS

Project size – measured by the work cost (in USD millions)	Success coefficient (in %)	Partial failure (in %)	Total failure (in %)	Total
Over 10	2	50	48	100
6 – 10	11	51	38	100
3 – 6	14	54	32	100
0,75 – 3	38	49	13	100
Under 0,75	71	24	5	100

Source: [25, p. 30].

TABLE X. THE EFFECTIVENESS OF APPLICATION SOFTWARE D&EP EXECUTION – WATERFALL VS. AGILE MODEL

Effectiveness	Waterfall model	Agile model	Average in 2010
Success coefficient (in %)	26	43	37
Partial failure (in %)	59	45	42
Total failure (in %)	15	12	21

Source: [2], [25, p. 21].

However, modernization variant recommended in the discussed case is not devoid of drawbacks though. Most of all, it evidently is not suitable for organizations where BSS had not functioned so far (in Poland approx. 95% of small companies do not use BSS – comparing to 50% in developed countries), for new organizations, new departments, and in case of fusion the modernization often ends in failure too. Moreover in modernization variant there are limited possibilities to implement fundamental business changes. What is more, the use of obsolete technologies is being continued, what makes cooperation with modern applications difficult, reduces usability, portability and maintainability of the modified application; performance is usually lower too. It is worth stressing that these attributes are the software product quality attributes of the ISO/IEC 9126 norm [30]. Thus, what appears to be open to doubt is reduction of costs and difficulties in maintaining the system as well as technological risk - this being one of the major goals of the solution variant to be chosen (see Section 2). It is also worth mentioning that the ISBSG data indicate lower productivity of such projects: in case of BSS D&EP consisting in developing new BSS from scratch it ranges on average from 9 (for 4GL) to 24.5 (for 3GL) work hours for developing 1 function point whereas in case of modernization projects it takes approx. 27 work hours on average to develop 1 function point [31].

VI. CONCLUSION AND FUTURE WORK

Based on the analysis of benchmarking data coming from the Standish Group repository, having been carried out with the use of *VirtualADVISOR* tool, it was concluded that what proves the best among the three BSS D&EP variants *in the discussed case* is variant consisting in modernization of application being used. Data analysis indicates that choosing the above mentioned variant is rational due to the criterion of both expected effectiveness and expected efficiency of project. This conclusion has been confirmed by the verification based on the repository of the SPR and ISBSG data, having been carried out with the use of *SPR Knowledge Plan* tool.

From the point of view of effectiveness and efficiency, modernization variant has many advantages yet it is not devoid of drawbacks though. What is more, this does not have to be the best solution in other cases, e.g., for real time systems, for small software product development/enhancement projects, or for organizations that specialise in developing specific kind of new software systems where there is possibility to use the already written code. It should be also mentioned that projects of higher risk, i.e., those having lower chance to succeed, often happen to be more efficient.

As indicated by the study results discussed in this paper, in view of exceptionally low effectiveness of BSS D&EP it is necessary to rationalize investment decisions being made with regard to such projects. To do so one may successfully use ever richer resources of benchmarking data having been collected in repositories with intention to support effective and efficient BSS D&EP execution. In the opinion of T.C. Jones: "For many years the lack of readily available benchmark data blinded software developers and managers to the real economics of software. Now (...) it is becoming possible to make solid business decisions about software development practices and their results (...). [Benchmarking – B.C.C.] data is a valuable asset for the software industry and for all companies that produce software" [32].

Appropriate benchmarking data most of all mean data pertaining to the type of software projects considered, being representative of this type. Undoubtedly the best solution is a situation when organizations use their own benchmarking data yet in practice it still happens that they rarely collect such data in a reliable and systematic manner, necessary to derive dependencies being specific to them. What reveals in this case is usefulness of repositories collecting general benchmarking data, created with improvement of software processes and products in mind. Repositories collecting general benchmarking data about software systems D&EP completed in the past include, apart from the information on mean values, also more precise data, dependent, among others, on the specificity of project and its product. Such repositories, which should be standardised according to the ISO/IEC 15939 norm [33], recent and representative of current technologies, may also support, among others (for more details see [15] and [34, pp. 3-4]):

- Proper software systems D&EP planning through:

- verification of the product requirements completeness,
- early and reliable estimation of the product size as well as project effort, cost and time,
- determining product size in a way so that it would ensure possibility of completing project on time and within the planned budget,
- determining optimum size of project team,
- finding balance among project attributes yet with priorities being taken into account (e.g., quality versus productivity),
- considering the outsourcing option,
- defining components of project environment,
- determining the influence of the chosen development tools and methods on the project,
- pricing of the product based on the cost per functional unit (i.e., function point - for more details see [12]).
- Proper management of the project risk through the verification of estimates for the project attributes reliability.
- Early and reliable control of project attributes throughout its accomplishment.
- Evolution of software D&EP organizations through possibility of:
 - comparing characteristics typical of given organization with characteristics in organizations having similar business profile (e.g., insurance, manufacturing, banking),
 - building organizational own database on project productivity,
 - increasing productivity of project activities,
 - reducing “time to market”, that is reducing time of developing and launching new products to the market.

On the other hand, this paper presented the possibility of using benchmarking data by a client in the rationalization of investment decision concerning the choice of the BSS D&EP execution variant, illustrated on the basis of a case study. Thanks to the rational investment decisions made on the basis of reliable and objective benchmarking data it is possible to reduce losses caused not only by abandoned projects but also by the large scale of overrunning the time and costs of BSS D&EP execution.

Collecting and analysis of the benchmarking data concerning software projects most of all is aimed to discover and understand regularities applying to various projects of this type. It will be possible only on the condition that repositories containing benchmarking data about software projects will continue to be extended – with particular emphasis put on these projects, which are characterised by the exceptionally low effectiveness, i.e., business software systems development and enhancement projects.

REFERENCES

- [1] B. Czarnacka-Chrobot, “Choosing a business software systems development and enhancement project variant on the basis of benchmarking data - case study”, Proc. of the 6th International Conference on Software Engineering Advances (ICSEA 2011), 23-28 October 2011, Barcelona, Spain, Luigi Lavazza, Luis Fernandez-Sanz, Oleksandr Panchenko, Teemu Kanstrén, Eds., International Academy, Research, and Industry Association, Wilmington, Delaware, USA, 2011, pp. 453-458.
- [2] Standish Group, “CHAOS manifesto 2011”, West Yarmouth, Massachusetts, 2011, pp. 1-48.
- [3] Standish Group, “CHAOS summary 2009”, West Yarmouth, Massachusetts, 2009, pp. 1-4.
- [4] B. Czarnacka-Chrobot, “The economic importance of business software systems size measurement”, Proc. of the 5th International Multi-Conference on Computing in the Global Information Technology (ICCGI 2010), 20-25 September 2010, Valencia, Spain, M. Garcia, J-D. Mathias, Eds., IEEE Computer Society Conference Publishing Services, Los Alamitos, California-Washington-Tokyo, 2010, pp. 293-299.
- [5] T. C. Jones, Patterns of software systems failure and success, International Thompson Computer Press, Boston, MA, 1995.
- [6] PCG, “2008 ERP report, topline results”, Panorama Consulting Group, Denver, 2008, pp. 1-2.
- [7] PCG, “2011 ERP report”, Panorama Consulting Group, Denver, 2011, pp. 1-15: <http://panoramaconsulting.com/Documents/2011-ERP-Report.pdf> (10.12.2012).
- [8] L. Tartanus and E. Kinczyk, “Projekty poza budżetem i harmonogramem” [“Projects going beyond budget and schedule”], Copmuterworld Poland, 22.11.2011; [\(http://www.computerworld.pl/artykuly/377447/Projekty.pozabudzetem.i.harmonogramem.html?utm_source=mail&utm_campaign=newsletter%20-%20Computerworld&utm_medium=Wiadomosci%20Computerworld%20\(html\)\)](http://www.computerworld.pl/artykuly/377447/Projekty.pozabudzetem.i.harmonogramem.html?utm_source=mail&utm_campaign=newsletter%20-%20Computerworld&utm_medium=Wiadomosci%20Computerworld%20(html)) (26.11.2011).
- [9] T. C. Jones, “Software project management in the twenty-first century”, Software Productivity Research, Burlington, 1999.
- [10] J. Johnson, “CHAOS rising”, Proc. of 2nd Polish Conference on Information Systems Quality, Standish Group-Computerworld, 2005, pp. 1-52.
- [11] Standish Group, “CHAOS summary 2008”, West Yarmouth, Massachusetts, 2008, pp. 1-4.
- [12] B. Czarnacka-Chrobot, “The economic importance of business software systems development and enhancement projects functional assessment”, International Journal on Advances in Systems and Measurements, vol. 4, no 1&2, International Academy, Research, and Industry Association, Wilmington, Delaware, USA, 2011, pp. 135-146.
- [13] Economist Intelligence Unit, “Global survey reveals late IT projects linked to lower profits, poor business outcomes”, Palo Alto, California, 2007: <http://www.hp.com/hpinfo/newsroom/press/2007/070605xa.html> (10.12.2012).

- [14] Standish Group, "The CHAOS report (1994)", West Yarmouth, Massachusetts, 1995; http://www.ics-support.com/download/StandishGroup_CHAOSReport.pdf (10.12.2012).
- [15] B. Czarnacka-Chrobot, "The role of benchmarking data in the software development and enhancement projects effort planning", in *New Trends in Software Methodologies, Tools and Techniques*, Proc. of the 8th International Conference SOMET'2009, H. Fujita, V. Marik, Eds., *Frontiers in Artificial Intelligence and Applications*, vol. 199, IOS Press, Amsterdam-Berlin-Tokyo-Washington, 2009, pp. 106-127.
- [16] A. Brown, "IS evaluation in practice", *The Electronic Journal Information Systems Evaluation*, vol. 8, no. 3, 2005, pp. 169-178.
- [17] E. Frisk and A. Plantén, "IT investment evaluation – a survey of perceptions among managers in Sweden", Proc. of the 11th European Conference on Information Technology Evaluation, Academic Conferences, 2004, pp. 145-154.
- [18] Z. Irani and P. Love, "Information systems evaluation: past, present and future", *European Journal of Information Systems*, vol. 10, no. 4, 2001, pp. 183-188.
- [19] S. Jones and J. Hughes, "Understanding IS evaluation as a complex social process: a case study of a UK local authority", *European Journal of Information Systems*, vol. 10, no. 4, 2001, pp. 189-203.
- [20] A. J. Silvius, "Does ROI matter? Insights into the true business value of IT", *The Electronic Journal Information Systems Evaluation*, vol. 9, issue 2, 2006, pp. 93-104.
- [21] B. Czarnacka-Chrobot, "Analysis of the functional size measurement methods usage by Polish business software systems providers", in *Software Process and Product Measurement*, A. Abran, R. Braungarten, R. Dumke, J. Cuadrado-Gallego, J. Brunekreef, Eds., Proc. of the 3rd International Conference IWSM/Mensura 2009, *Lecture Notes in Computer Science*, vol. 5891, Springer-Verlag, Berlin-Heidelberg, 2009, pp. 17-34.
- [22] Standish Group, "Modernization – clearing a pathway to success", West Yarmouth, Massachusetts, 2010, pp. 1-16.
- [23] Software Productivity Research: <http://www.spr.com/spr-knowledgeplanr.html> (10.12.2012).
- [24] ISBSG, "Data demographics release 11", International Software Benchmarking Standards Group, Hawthorn, Australia, June 2009, pp. 1-24.
- [25] Standish Group, "The CHAOS manifesto", West Yarmouth, Massachusetts, 2009, pp. 1-54.
- [26] M. Cohn, *Agile estimating and planning*, Prentice Hall Professional Technical Reference, Upper Saddle River, NJ, 2006.
- [27] ISBSG, "Techniques & tools – their impact on projects", International Software Benchmarking Standards Group, Hawthorn, Australia, 2010, pp. 1-7.
- [28] T.C. Jones, *Applied software measurement: global analysis of productivity and quality*, 3rd edition, McGraw-Hill Osborne Media, 2008.
- [29] ISBSG, "Software project characteristics or events that might impact development productivity", International Software Benchmarking Standards Group, Hawthorn, Australia, 2007, pp. 1-2.
- [30] ISO/IEC 9126 Software Engineering – Product Quality – Part 1-4, ISO, Geneva, 2001-2004.
- [31] Ch. Symons, "The performance of real-time, business application and component software projects", Common Software Measurement International Consortium (COSMIC) and ISBSG, September 2009, pp. 1-45.
- [32] ISBSG: <http://www.isbsg.org> (10.12.2012).
- [33] ISO/IEC 15939 Systems and software engineering -- Measurement process, ISO, Geneva, 2007.
- [34] *Practical Project Estimation (2nd edition): A toolkit for estimating software development effort and duration*, P.R. Hill, Ed., International Software Benchmarking Standards Group, Hawthorn, Australia, 2005.

Mining Test Cases: Optimization Possibilities

Edith Werner* and Jens Grabowski†

*Neumüller Ingenieurbüro GmbH, Nürnberg, Germany
edithbmwerner@googlemail.com

†Software Engineering for Distributed Systems Group,
Institute for Computer Science, University of Göttingen, Göttingen, Germany
grabowski@cs.uni-goettingen.de

Abstract—System monitors need oracles to determine whether observed traces are acceptable. One method is to compare the observed traces to a formal model of the system. Unfortunately, such models are not always available — software may be developed without generating a formal model, or the implementation deviates from the original specification. In previous work, we have proposed a learning algorithm to construct a formal model of the software from its test cases, thereby providing a means to transform test cases for offline testing into an oracle for monitoring. In this paper, we refine our learning algorithm with a set of state-merging rules that help to exploit the test cases for additional information. We discuss our approach in detail and identify optimization areas. Using the additional information mined from the test cases, models can be learned from smaller test suites.

Keywords-Machine Learning; Reverse Engineering; Testing

I. INTRODUCTION

Today, software systems are generally designed to be modular and reusable. A common scenario of a modular, reusable system is a web service, where simple services are accessed as needed by various clients and orchestrated into larger systems that can change at any moment. While the vision of ultimate flexibility is clearly attractive, there are also drawbacks, as the further usage of a module is difficult to anticipate. In this scenario, it may be advisable to monitor a system for some time after its deployment, to detect erroneous usage or hidden errors.

Monitors are used to observe the system and to assess the correctness of the observed behavior. To this end, monitors need oracles that accept or reject the observed behavior, e.g., a system model that accepts or rejects the observed traces of the monitored system. Unfortunately, the increasing usage of dynamic software development processes leads to less generation of formal models, as the specification of a formal model needs both time and expertise. Generating a formal model in retrospect for an already running system is even harder, as the real implementation often deviates from the original specification.

We propose a method for learning a system model from the system's test cases without probing the System Under Test (SUT) itself [1]. When test cases are available, they often are more consistent to the system than any other model. Ideally, they take into account all of the system's possible

reactions to a stimulus, thereby classifying the anticipated correct reactions as accepted behavior and the incorrect or unexpected reactions as rejected behavior. As the test cases are developed in parallel to the software, they provide a means to judge the correct behavior of the system. Also, test cases are generated at different levels of abstraction, e.g., for unit testing, integration testing, and system testing. By selecting the set of test cases to be used, the abstraction level of the generated model can be influenced.

The basis of our approach is a learning algorithm, first introduced by Angluin [2], which learns a Deterministic Finite Automaton (DFA). To learn from test cases, we adapted the query mechanisms of the algorithm [3]. Experiments with our approach show that while a model can be learned this way, the algorithm only accepts simple traces as input, thereby losing additional information from the test cases, e.g., regarding branching, default behavior, or synchronization. We believe that exploitation of this additional information would enhance the learning algorithm.

In this paper, we propose a state-merging approach, termed *semantic state-merging*, which exploits the semantic properties of test cases in order to identify implicitly defined behavior. We first define a data structure, the *trace graph*, to store the available test cases. Then, we define merging rules for cyclic test cases and for test cases with default branches for the construction of the trace graph. Based on the experiences gathered through a prototypical implementation, we identify optimization areas and possible solutions.

The remainder of this paper is structured as follows. Section II gives an overview on related work. In Section III, we introduce the foundations of our work in testing and machine learning. Section IV describes the trace graph and its construction. Based on this, Section V defines our approach to semantic state-merging on test cases. Subsequently, in Section VI, we give an overview on our experimental results. Section VII discusses the learning approach and describes solution ideas to open questions. In Section VIII, we conclude with a summary and an outlook.

II. RELATED WORK

As our approach combines learning techniques and state merging, we need to take into account related work from

both areas. In the following, we give an overview on relevant articles regarding the adaptation of Angluin's learning algorithm and state-merging and establish the differences to our own work.

A. Related Work in Learning

During the last years, a number of approaches have adapted Angluin's learning algorithm in combination with testing. Mainly, the approaches focus on the learning side of the problem and refine the properties of the generated model. Among the most recent adaptations are approaches to learning Mealy machines [4] and parameterized models [5], [6], [7], [8]. Some approaches can handle large or even infinite message alphabets [5] or potentially infinite state spaces [6]. In all those approaches, the learning algorithm generates test cases that are subsequently executed against the SUT, so that the System Under Test itself is the oracle for the acceptability of a given behavior.

Some approaches use outside guidance to improve the learning approach. The algorithm presented in [9] learns workflow petri nets from event logs and handles incomplete data by asking an external teacher. In [10], learning is used in a modeling approach. In this approach, a domain expert provides Message Sequence Charts representing desired and unwanted behavior.

Our approach differs from the above in two aspects. First, we aim at generating a model for online monitoring. To this end, we need a model that is independent from the implementation itself. Therefore, we can neither use the implementation as an oracle nor learn from event traces generated by the implementation. Instead, we choose to learn from a test suite that was developed due to external criteria. Using a test suite also leads to the second difference of our approach. Where other approaches rely on unstructured data, a test suite provides relations between the distinct traces. We exploit those relations in order to enhance our learning procedure. Where other approaches address the learning side of the problem, our focus is actually on the structure of the teacher.

B. Related Work in Stage-Merging

The basic idea of the state-merging approach is to analyze examples of the target automaton, identify possible states, and merge similar states until the remaining states are considered to be sufficiently distinct. The notion was first introduced by Biermann [11], who used it to generate computer programs from short code samples. Meanwhile, the merging techniques have been extended to logical evaluation of the samples [12], [13] and different heuristics have been introduced [14].

Also, different input domains have been explored. One the one hand, there are approaches that synthesize models from partial models like scenario diagrams [15], [16], [17], [18]. On the other hand, there are approaches that reconstruct

a behavioral model of an existing software by merging observed traces [19], [20], [21].

In all cases, the main problem in state-merging is over-generalization, i.e., a false merging of states, thereby overly simplifying the model. In our approach, we derive the merging rules from the semantic context of our domain, i.e., the properties of the test specification language. This leads to a more conservative merging and avoids erroneous mergings, while nevertheless enlarging the sample space sufficiently.

III. FOUNDATIONS

In the following, the foundations of testing and on the learning of DFA are presented.

A. Testing

A test case is itself a software program. It sends stimuli to the SUT and receives responses from the SUT. Depending on the responses, the test case may branch out, and a test case can contain cycles to test iterative behavior. To each path through the test case's control flow graph, a verdict is assigned. A common nomenclature is to use the verdict **pass** to mark an accepting test case and the verdict **fail** to mark a rejecting test case. An *accepting* test case is a test case where the reaction of the SUT conforms to the expectations of the tester. This can also be the case, when an erroneous input is correctly handled by the SUT. Accordingly, a *rejecting* test case is a test case where the reaction of the SUT violates its specification. Depending on the test specification, there may be additional verdicts, e.g., the Testing and Test Control Notation version 3 (TTCN-3) [22] extends the verdicts **pass** and **fail** with the additional verdicts **none**, **inconc**, and **error**: **none** denotes that no verdict is set; **inconc** indicates that a definite assessment of the observed reactions is not possible, e.g., due to race conditions on parallel components; and **error** marks the occurrence of an error in the test environment. During the execution of a test case, the verdict may be changed at different points. The overall assessment of a test case depends on the verdicts set along the execution trace, and is computed according to the rules of the test language. E.g., in TTCN-3, the overall verdict may only be downgraded, i.e., once an event was rated as **fail** the overall verdict may not go back to **pass**. For most SUTs, there is a collection of test cases, where each test case covers a certain behavioral aspect of the SUT. Such a collection of test cases for one SUT is called a *test suite*.

The main objective when constructing test cases for a software system is to assure that the specified properties are present in the SUT. To test against a formal specification, e.g., in the form of a DFA, test cases are derived from the model by traversing the model so that a certain coverage criterion is met, e.g., *state coverage* or *transition coverage*. State coverage means that every state of the model is visited by at least one test case. Transition coverage means that every transition of the model is visited by at least one test

case. The largest possible coverage of a system model is *path coverage*, where every possible path in the software is traversed.

B. Learning a Finite Automaton Model from Test Cases

Our learning approach is based on a method proposed by Angluin [2]. The algorithm consists of the *teacher*, which is an oracle that knows the concept to be learned, and the *learner*, who discovers the concept. The learner successively discovers the states of an unknown target automaton by asking the teacher whether a given sequence of signals is acceptable to the target automaton. To this end, the teacher supports two types of queries. A *membership query* evaluates whether a single sequence of signals is a part of the model to be learned. An *equivalence query* establishes whether the current hypothesis model is equivalent to the model to be learned.

For learning from test cases, we need to redefine the two query types in relation to test cases. The most important mechanism of the learning algorithm is the membership query, which determines the acceptability of a given behavior. In our case, the behavior of the software and thus of the target automaton is defined by the test cases. Since the test cases are our only source of knowledge, we assume that the test cases cover the complete behavior of the system. In consequence, we state that every behavior that is not explicitly allowed must be erroneous and therefore has to be rejected, i.e., $rejected \equiv \neg accepted$. Accordingly, we accept a sequence of signals if we can find a **pass** test case matching this sequence, and reject everything else.

The equivalence query establishes conformance between the hypothesis model and the target model. This is exactly what a test suite is designed for, therefore, we redefine the equivalence query as an execution of the test suite against the hypothesis model, where every test case in the test suite must reproduce its verdict. A detailed description of the learning algorithm can be found in [3], [23].

IV. REPRESENTING TEST CASES

For the learning procedure, it is important that queries can be answered efficiently and correctly. Therefore, we need a representation of the test suite that is easy to search and provides a means to compactly store a large number of test cases. In the following, we define the *trace graph* as a data structure and describe its construction.

A. The Trace Graph

As described in Section III-A, a test case is itself a piece of software and can therefore be represented as an automaton containing a number of event sequences. Usually, a test case distinguishes events received from the SUT, events sent to the SUT, and internal actions like value computation or setting verdicts. Each possible path through the test case must contain the setting of a verdict.

For the learning procedure, we only regard input and output events as the transitions in our target model and ignore internal actions except for the setting of verdicts. The verdicts are used to identify accepting test cases.

In general, every test case combines a number of traces, depending on the different execution possibilities. At the same time, a test suite contains a number of test cases, where different test cases may contain identical traces as they partly overlap. To present the test cases to the learning algorithm, we combine all traces from all test cases in the test suite into a single data structure, the *trace graph*, thereby eliminating duplicates and exploiting overlaps.

To enable an efficient search on the test cases, the trace graph is based on a labeled search tree, where all traces share the same starting state. Traces with common prefixes share a path in the trace graph as long as their prefixes match. For the state-merging approach, the nodes in the trace graph are annotated with the verdicts. Cycles in the test cases are represented in the trace graph by routing the closing edges back to the starting node of the cycle. For better control, nodes where a cycle starts are also marked.

The trace graph forms the basic data structure for our semantic state-merging. The semantic state-merging methods depend on the information contained in the test cases, which in turn depends on the test language. To represent this, the trace graph can be extended to represent diverse structural information on the test cases by defining additional node labels. That way, information on the test cases will only affect the construction of the trace graph, but not the learning procedure that depends on its structure.

B. Constructing the Trace Graph

To construct the trace graph, we dissect the test cases into single traces and add them to the trace graph. Starting in the root of the trace graph, the signals in the trace to be added are matched to the node transitions in the trace graph as far as possible. We call this part of the trace the *common prefix*. The remainder of the new trace, the *postfix*, is then added to the last matched node. Algorithm 1 describes the procedure in pseudo code.

Cycles of the test case automaton need special treatment, as a cycle means that an edge loops back to an existing node. To this end, we separate the cyclic traces into three parts, a prefix leading into the cycle, the cycle itself and a postfix following the cycle. We then add the prefix and the cycle, whereby the last transition in the cycle is linked back to the beginning of the cycle. Finally, the postfix then is added to the trace graph.

C. Querying on the Trace Graph

The most important mechanism of the learning algorithm is the membership query, which determines the acceptability of a given behavior. In our case, the behavior of the software and thus of the target automaton is defined by the test cases.

Data: A sequence of signals w

```

1 Start at the root node  $n_0$  of the trace graph;
2 for all signal in  $w$  do
3   Get the first signal  $b$  in  $w$ ;
4   if the current node has an outgoing edge marked  $b$ 
   then
5     Move to the  $b$ -successor of  $n$ , which is  $\delta(n, b)$ ;
6     Remove the first signal from  $w$ ;
7   else
8     // The signal is unknown at the current node
9     Add  $w$  as a new subgraph at the current node;
10    return;
11 end

```

Algorithm 1: Add a Trace to the Trace Graph

Posing a membership query against the trace graph amounts to searching the queried trace in the graph and computing the verdict of the trace. The computation of the verdict has to be adapted to the semantics of the test specification language. In TTCN-3, the policy is that a verdict can only get worse. Accordingly, the overall verdict of a trace is the worst verdict that is stored in the nodes belonging to that trace. In other words, for a trace to be accepted, there must be at least one **pass** verdict and no **fail** verdicts stored in the trace graph for that trace. We rate any trace that is not found on the trace graph as not acceptable and therefore apply the verdict **fail**. The same policy applies to incomplete traces, i.e., the queried trace does not end in a final state or no verdict has been applied during the trace.

The purpose of the equivalence query is to prove that the hypothesis automaton conforms to all test cases in the test suite. This can be regarded as a structural test of the hypothesis automaton against the test suite. Based on the trace graph, the equivalence query is realized as a tree walking algorithm on the trace graph, where the number of cycle expansions is registered and the generated traces are recorded to keep track of interleaved cycles. As short counter-examples provide better results in the learning algorithm, a simple depth-first tree walking algorithm is not sufficient. To extract the shortest possible counter-example in each iteration, we need to use an iterative deepening search. The complexity of such an iterative deepening search depends on the branching factor of the tree to be searched. For the trace graph, the branching depends on the structure of the test suite.

V. MINING THE TEST CASES

So far, the state-merging in the trace graph only means the combination of the test case automata, where traces are only merged as far as their prefixes match. The trace graph therefore exactly represents the test cases, but nothing more. In the following, we show two techniques to derive

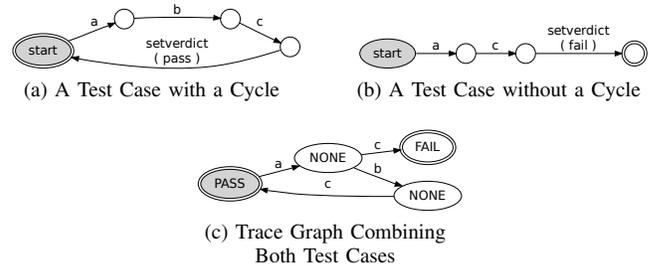


Figure 1. Precedence of Cyclic Behavior

additional traces based on our knowledge of test cases.

A. Cycles and Non-Cycles

When testing a software system with repetitive behavior or a cyclic structure, the cycle has of course to be tested. However, usually it is sufficient to test the correct working of the cycle in one test case. In all other test cases the shortest possible path through the software is considered, which may mean that test cases execute only a part of a cycle or completely ignore a cycle. Depending on the test purpose, the existence of the cycle might not even be indicated in the test case. As long as the cycle itself is tested by another test case, the test coverage is not influenced. This approach results in shorter test cases, which means shorter execution time and thus faster testing. Furthermore, the readability of the test cases is increased. While the preselection of possible paths for cycles is appropriate for software testing, for machine learning it is desirable to have access to all possible paths of the software.

Consider the two test cases shown in Figures 1a and 1b. Although this is only a small example for demonstration purposes, the setting is quite typical. The test case shown in Figure 1a tests the positive case, that is, a repeated iteration of the three signals a , b , and c . The test case shown in Figure 1b tests for a negative case, namely what happens if the system receives the signal c too early. In the latter test case, the repetitive behavior is ignored, as it has been tested before and the test focus is on the error handling of the system. However, usually this behavior could also be observed at any other repetition of the cycle.

For the learning procedure, we would like to have all those possible failing traces, not only the one specified. We therefore define a precedence for cycles, which means that whenever a cycle has the same sequence of signals as a non-cyclic trace, the non-cyclic trace is integrated into the cycle. Figure 1c shows the trace graph combining the two test cases in Figures 1a and 1b. Besides the trace a, c , **setverdict(fail)** explicitly specified in Figure 1b, the trace graph also contains traces where the cycle is executed multiple times, $(a, b, c)^*$, a, c , **setverdict(fail)**. With precedence of cycles, the test suite used as input to the learning algorithm can be more intuitive, as cycles only need to be specified once.

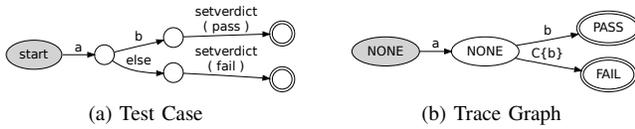


Figure 2. Representing Default Branches

B. Default Behavior

Another common feature of test cases is the concentration on one test purpose. Usually, the main flow of the test purpose forms the test case, while unexpected reactions of the SUT are handled in a general, default way. Still, there may exist a test case that tests (a part of) this default behavior more explicitly.

Default branches usually occur when the focus of the test case is on a specific behavior, and all other possible inputs are ignored or classified as **fail**. Also, sometimes a test case only focuses on a part of the system, where not all possible signals are known. In such cases, the test case often contains a default branch, which classifies what is to be done on reading anything but what was specified.

For our application, this poses two challenges. The first challenge is in the learning procedure. For the different queries, we need to have as many explicitly classified traces as possible, but at the same time we do not want to blow up the size of the test suite. The second challenge is in the construction of the trace graph. When adding all different traces into one combined structure, the implicit context of what is “default” in the local test case is lost. Also, sometimes another test case uses the same default, adds more specific behavior in the range of the default, or defines a new default that slightly differs. We therefore need a method of preserving the local concept of “default” in the test cases and a method of combining different defaults in the trace graph.

Consider a typical default situation, like a **default** statement in a **switch-case** environment. The **default** collects all cases that are not explicitly handled beforehand. As branching on alternatives splits the control flow in a program, each of the branches belongs to a different trace. Therefore, when taking the traces one by one, the context of the default is not clear. To preserve this context, instead of **default** we record the absolute complementary of the set of other alternatives, which is $\mathbb{C}\{a, b\}$. A *complementary set* is a set that contains everything but the specified elements. Figure 2 shows a test case with defaults (Figure 2a) and its representation as a trace graph using the complementary set notation (Figure 2b). The branch marked with $\mathbb{C}\{a\}$ represents every branch not marked with a .

Figure 3 shows a trace graph with a default branch in a general way. There are some arbitrary transitions leading to the default (marked with *prefix*), the default branching itself with an edge marked a and an edge marked $\mathbb{C}\{a\}$

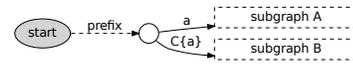


Figure 3. Generic Trace Graph with Default Branch

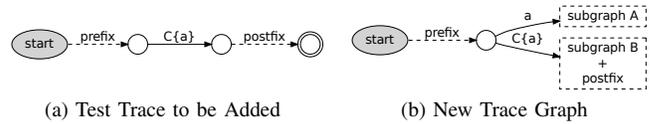


Figure 4. Add a Trace with a Matching Default

(“everything but a ”), and the arbitrary subgraphs of a and $\mathbb{C}\{a\}$.

When adding a trace with a matching prefix to this trace graph, the signal s following the prefix can be matched to the trace graph according to one of the following three cases.

- *Exact Match*: s matches one of the branches of the trace graph, i.e., if s is a complementary set, it is identical to the complementary set in the trace graph.
- *Subset*: s matches one signal (or a subset of signals) of the complementary set in the trace graph.
- *Overlap*: s is a complementary set, and overlaps the complementary set in the trace graph.

The first and simplest case is the *exact match*, where a trace with a matching complementary set is added. As the complementary sets are identical, it suffices to add the postfix of the trace to the subgraph of the default already in the trace graph. Figure 4 illustrates this. Figure 4a shows the test trace to be added. The prefix of the trace matches the prefix of the trace graph (see Figure 3) and the complementary set $\mathbb{C}\{a\}$ matches the complementary set in the trace graph. Therefore, the postfix of the trace has to be added to the subgraph of the complementary set. Assuming that there are no other defaults in the postfix, this is done according to the construction rules described in Section IV-B. Figure 4b depicts the resulting trace graph after the new trace was added.

In the second case, the new trace matches a *subset* of

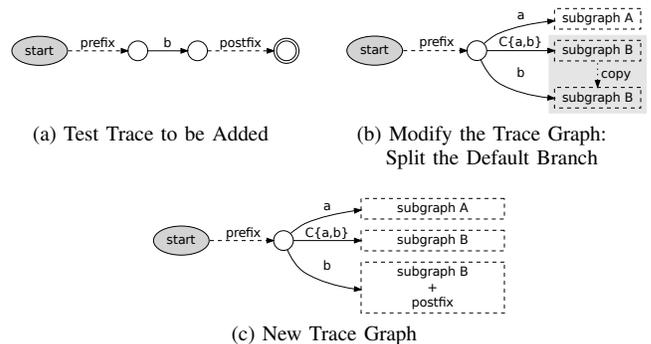


Figure 5. Add a Trace with a Subset of the Default

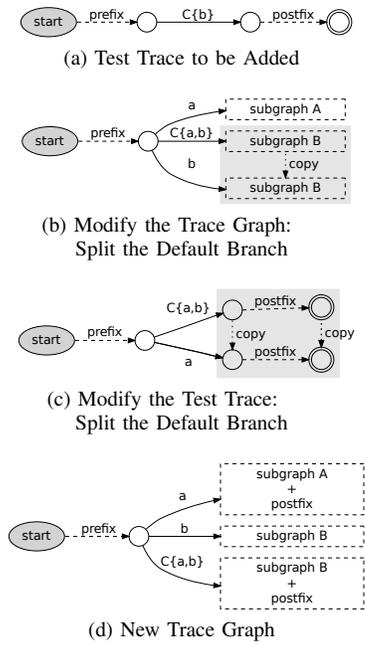


Figure 6. Add a Trace with a Differing Default

the complementary set in the trace graph. The situation is depicted in Figure 5, the signal following the prefix in the trace (Figure 5a), b , is a subset of the complementary set $\mathbb{C}\{a\}$. However, the postfix cannot simply be added to the subgraph of the complementary set, as this would allow unspecified traces. Instead, before adding the postfix, the trace graph is modified as shown in Figure 5b. The signal b is removed from the complementary set and represented by a distinct edge. Now, the new trace matches exactly and the adding proceeds as described for the first case. Figure 5c shows the result.

In the third and last case, the complementary sets of the new trace and the trace graph *overlap* (see Figure 6). The trace contains an edge marked with the complementary set $\mathbb{C}\{b\}$ (Figure 6a), whereas the trace graph contains an edge marked with the complementary set $\mathbb{C}\{b\}$ (see Figure 3). The complementary set of the test trace to be added does not fit the complementary set of the trace graph, but there is an overlap, i.e., every signal which is neither a nor b matches both sets.

The solution is similar to the second case. The transitions in the trace need to match the transitions in the trace graph, so the sets are split accordingly. For the trace graph, the edge marked b is branched out from the complementary set (Figure 6b). The remaining complementary set in the trace graph is $\mathbb{C}\{a, b\}$. However, the complementary set of the test trace still does not match, so the test trace is also split (Figure 6c). The complementary sets of the trace graph and the test trace are now identical, $\mathbb{C}\{a, b\}$, but the test trace has been split into two test traces. Now, the two resulting test traces can be added to the trace graph, resulting in the

trace graph shown in Figure 6d.

The described techniques also generalize to sets with more than one element. In this case, the sets associated with the split branches are determined as the intersections and differences of the given sets.

VI. IMPLEMENTATION AND CASE STUDY

To assess the power of our learning approach, we have developed a prototypical implementation [23]. The implementation realizes an Angluin-style learner, which is adapted to learning from test cases, and the organization of the test data into a trace graph as discussed in Sections IV and V. Using the prototype, we performed a case study based on the *conference protocol* [24]. The conference protocol describes a chat-box program that can exchange messages with several other chat-boxes over a network.

In the following, we will give a short overview of the prototypical implementation. Subsequently, we describe the experiments that were performed with two versions of the conference protocol. In the last section of this chapter, we will compare the two experiments and draw some conclusions.

A. Prototypical Implementation

Our prototype is implemented in the programming language Java, the abstract structure is shown in Figure 7 as a Unified Modeling Language (UML) class diagramm.

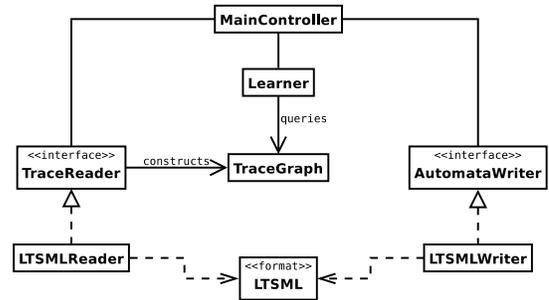


Figure 7. Abstract Structure of the Implementation

The main classes of our prototype implementation are explained further in the following. The class `Learner` implements Angluin’s learning algorithm. In every iteration of the learning algorithm, a new counter example is obtained via an equivalence query and used to detect a new state. The discovered states are organized in a classification tree, which is also used to generate the new hypothesis automaton. The two queries, equivalence query and membership query, are implemented according to our adaptation to learning from test cases, and mapped onto a trace graph structure.

In the class `TraceGraph`, the basic methods of semantic state-merging are implemented. A trace graph structure is constructed by adding traces from test cases. The precedence of loops (Section V-A) is currently implemented implicitly,

as loops are simply added first to the trace tree. Default branches (Section V-B) are not yet implemented. In consequence, the currently implemented `TraceGraph` and `Learner` classes are generic and can be used for any test specification language.

For the input of the test cases and the output of the hypothesis automaton, generic interfaces were defined. In our prototype, both interfaces are implemented using the LTSML format that can be used to represent any type of automaton [25]. For the test cases, we focus on the test specification language TTCN-3, i.e., the `TraceReader` recognizes TTCN-3 keywords and generates traces according to the semantics of TTCN-3.

B. The Conference Protocol

Our case study is based on a chat-box program described in [24], the *conference protocol*. We have adapted the protocol to the constraints of our learning procedure.

The conference protocol describes a chat-box program that allows users to participate at a conference chat over a network.

- A user enters an existing conference by sending the service primitive *join*.
- Then, the user can send messages to the conference chat (*datareq*) and receive messages from the conference chat (*dataind*). Each *datareq* causes *dataind* messages to be issued to all other participating users and vice versa.
- At any time after a *join*, the user can leave the conference by sending the service primitive *leave*.

The chat-box program converts each service primitive into a protocol data unit, which is then sent to each of the other participating chat-boxes over a network. Figure 8 illustrates this scenario in a UML interaction diagram.

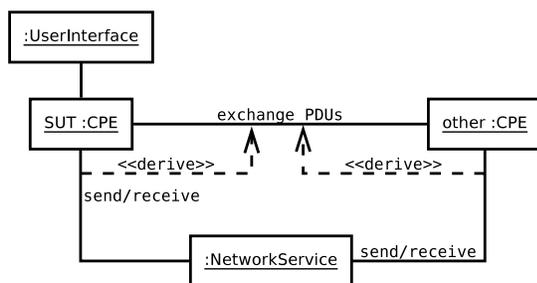


Figure 8. Two CPEs Connected over a Network Service

C. Mining for Cycles

In the first experiment, we want to assess our approach to mining the test cases for cycles. To limit the complexity of the model, we assume that the chat-boxes send messages to each other in a fixed sequence. Based on this assumption, we generate two test suites.

In the first test suite, we build the cases to satisfy a boundary-interior coverage, where the cycles in the data transmission phase of the protocol are executed once or twice, or skipped. The trace graph generated for this test suite contains no cycles. In the second test suite, we explicitly declare cycles, instead of unrolling them.

Table I shows our results for this experiment. The protocol was scaled according to the number of participating chat-boxes. As the table shows, the semantic state-merging of cycles reduces the size of the trace graph by more than half in this example, while the learned automaton was identical. Also, the test suite can be smaller. In addition, the compact version of the trace graph also allows an optimized equivalence query.

D. Limits in Learning Complex Communication

In order to assess the limits of our approach, we extend our version of the conference protocol. We now assume that the network service may mix up the signals, so that the data units are observed in an arbitrary sequence. In our test scenario, we want to accept all traces, where a chat-box correctly joined and left the conference. This means that all participating chat-boxes have received the *join* messages before the first data packages occur, and that no data packages occur after the *leave* message has been sent.

Every service primitive is distributed to $n - 1$ other chat-boxes, which means $(n - 1)!$ correct paths for joining and leaving the conference and also for sending data. In addition, there are $n - 1$ correct paths to receive data. We explicitly specify the data transfer as cyclic behavior. Therefore, we can compute the number of correct traces as $((n - 1)!)^3 * (n - 1)$, or $(|join|) \cdot (|send|) \cdot (|leave|) \cdot (|receive|)$, where $|service\ primitive|$ denotes the number of correct sequences for the services primitive.

The goal of this experiment is to find out how many test cases are needed to correctly learn the protocol. We tried different approaches to generate test cases for this version of the conference protocol.

A common coverage criterion in testing is the branch coverage, where every branch of the SUT is executed. In application to the conference protocol, this means that we have to cover every serialization of data units. However, it turned out that the learned automata do not correctly represent the intended protocol. A closer look at the model reveals that the learned automaton contains the traces exactly as they were specified in the test cases. Instead of generalizing from the input data, the learning algorithm learned every input trace by heart. This effect could be reproduced with different versions of the test suite. Only by using a test suite satisfying path coverage of the expected automaton, we could learn the correct automaton.

We deduce that the structure of the SUT has an influence on the complexity of the learning process and that for correct machine learning, the test suite has to be as complete as

Number of Chat-Boxes	Size of Target Automaton	Size of the Trace Graph		Size of the Test Suite	
		Without Cycles	With Cycles	Without Cycles	With Cycles
1	72 edges, 8 nodes	33 nodes	13 nodes	6 pass traces	2 pass traces
2	168 edges, 12 nodes	60 nodes	22 nodes	9 pass traces	3 pass traces
3	304 edges, 16 nodes	90 nodes	30 nodes	12 pass traces	4 pass traces
4	480 edges, 20 nodes	120 nodes	40 nodes	15 pass traces	5 pass traces
5	696 edges, 24 nodes	164 nodes	48 nodes	18 pass traces	6 pass traces

Table I
EFFECT OF SEMANTIC STATE-MERGING

possible. In fact, this precondition on the learning sample has been described before as the need for a “structurally complete” sample. As a rule of thumb, we might say that “the larger the test suite, the smaller the automaton”, as a large test suite usually allows more possible paths.

Experimentation has also shown that the growing size of the test suite affects our learning procedure in two related points. The first is the generation of the trace graph. For larger test cases the preprocessing steps, such as cycle detection, are harder to handle. The second is the equivalence query, where the whole test suite has to be executed again and again. Interestingly, it is not necessary to compute all interleaving cycle executions for the equivalence query. Instead, the crucial points for the equivalence query turn out to be the intersections between different paths through the SUT. Therefore, for a better scalability of our learning procedure, we should aim at detecting intersections of the test cases in the trace graph, thereby minimizing the trace graph and reducing the necessary size of the test suite while keeping its expressiveness.

VII. DISCUSSION AND OPEN QUESTIONS

While our experiments proved the suitability of the learning approach, they also raised a number of questions regarding specific properties of the used data structures and the algorithm. Some of the observations confirmed design decisions, while at other points, decisions turned out to be less than optimal. The following sections provide an assessment of the parts of our learning approach. We reassess the generated automaton, the use of a test suite as input sample, and the learning algorithm itself with respect to their suitability for our purposes. As some of the encountered questions have also attracted the attention of other researchers, there are a number of possible solutions available that could be adapted to our learning approach. In other situations, a number of possible solutions are suggested that have not been investigated yet.

A. Suitability of the Learned Automaton

Angluin’s learning algorithm generates a plain DFA, which consists of a set of states partitioned into accepting and rejecting states, an input alphabet triggering state transitions and a corresponding state transition relation. As we have argued in [3], this type of automaton is suitable

for representing system models. The drawback of DFAs is that to express the same information as a more advanced model, more states are needed, making the automaton large. Contrary to expectations, it was not the size of the target automaton that proved to be a problem, but its structure and the repercussions on the size of the test suite needed to correctly identify the automaton.

1) *Influence of Parameters:* A DFA representing a parameterized process such as the conference protocol described in Section VI-B contains a number of paths to cover different serializations of the parameters. The experiments show that to correctly learn all the different serializations, not only all of them need to be represented in the learning sample, but also in every possible combination. This also correlates to the results by Berg et al. regarding prefix-closed models [26], which state that prefix-closed automata are harder to learn than random automata, as the learning algorithm would need to perform a membership query for every prefix.

Berg et al. address this problem by proposing an approach to learn parameterized automata [27]. Based on the original version of Angluin’s learning algorithm, which uses an observation table to store the discovered information, they introduce a guard-labeling on the entries of the table, describing an input partitioning. Then, the result of an equivalence query can also be the splitting of a partition beside the discovery of a new state or the acceptance of the learned automaton. A similar approach is described by Li et al. [28], which has the advantage of taking into account both input and output signals, where Berg et al. only consider input signals.

Both proposed solutions for learning parameterized models rely on the original version of Angluin’s learning algorithm, which uses an observation table to store the information learned, and the table format is essential in computing the parameterization. In contrast, our approach to learning from test cases uses a variation introduced by Kearns and Vazirani [29], which stores the gathered information in a classification tree. Therefore, an adoption of those solutions requires some adaptations.

2) *Handling Non-Applicable Signals:* Another structural problem of the learned DFA is that the learning algorithm always generates a fully specified automaton, i.e., an automaton where in every state, a target state for every possible signal is specified. As the essence of state based systems

is that the applicable actions depend on the state of the system, most states of an automaton can only process a subset of the global signal alphabet. The handling of the non-applicable signals then depends on the semantics of the system. One commonly adopted approach is that the system should robustly ignore all unspecified signals, which implies that unspecified transitions at a given state are treated as self-loops.

However, this approach cannot be adopted by Angluin's learning algorithm, as the algorithm only discerns accepted and rejected traces and therefore cannot tell whether a signal is not specified and should be ignored via a self-loop or whether a signal is explicitly rejected and should lead to a fail state. In consequence, the learning algorithm routes all unspecified or rejected signals into a global fail state, thereby generating a fully specified automaton that rejects non-applicable signals.

Due to the properties of the learning algorithm, we can identify the global fail state in the learned automaton, as it is the first rejecting state discovered. Therefore, it would be possible to remove the global fail state and to replace transitions leading into it by self-loops to their source states. This is, however, not a safe transformation, as thereby all explicitly failing transitions would also be transformed into self-loops. In consequence, to learn a DFA that ignores some inopportune signals, those self-loop transitions need to be explicitly specified in the test suite. This obviously leads to a larger test suite, which is also less intuitive and less readable. Alternative approaches would be to distinguish explicitly and implicitly rejected transitions during learning, generating self-loops for implicitly rejected transitions, or to implement a smart transformation algorithm that checks transitions to the global fail state before removing them.

B. Suitability of a Test Suite as Sample Data

The main idea of our learning approach was to use a test suite as input data, as the verdicts **pass** and **fail** readily provided an assessment of acceptable and rejectable system traces. While this assumption holds true, mapping test cases to input traces of Angluin's learning algorithm nevertheless reduces the expressiveness of the test cases considerably. The test cases need to be linearized, a common starting state has to be established, and all circumstantial information as parameters and ports have to be integrated into the input of the target automaton. Especially the flattening of parameters and ports leads to an exponential blowup of the number of test case traces.

While Angluin's algorithm depends on traces, the semantic state-merging approach is designed to exploit the test language specific properties of test cases. The test language specific back-end then represents the sample data in a generic way to the learning algorithm. This way, the learning algorithm provides a common front-end to be combined with different test language specific back-ends. In consequence,

further optimization regarding the representation of the test suite mainly concerns the state-merging part of our hybrid algorithm.

1) *Mining Additional Properties:* The state-merging techniques introduced in Section V define how to merge traces by generating a prefix tree, the representation and handling of cycles in the trace graph and the handling of default branches. However, cycles and defaults are only the most common properties of test languages.

Stable testing states define known and checkable states of the SUT. By marking the according states in the trace graph, a test case containing a marked testing state could be directly connected at the given state. Thereby, the need for a common starting state could be avoided.

Parallel behavior can be explicitly defined, especially in test languages that are targeted at distributed testing. By defining an according operator on the trace tree, membership queries containing different sequentializations of parallel behavior could be answered correctly without explicitly representing every such path in the test suite.

Besides the verdicts **pass** and **fail**, some test languages define additional verdicts assigned on inconclusive behavior or on an error in the test environment. In the current learning approach, everything not accepted, i.e., assigned a **pass** verdict, is rejected. However, in an open world approach, the answer "I don't know" could be given by the membership oracle. In this case, the mapping of the test verdicts has to be reconsidered. The verdict **inconc**, which is used by TTCN-3 to indicate that the result of the test case cannot be decided, maps naturally on an "I don't know" for the learner—the teacher does not know whether the behavior is acceptable or not. Then again, the verdict **error** is considered by TTCN-3 to be more severe than a verdict **fail**, but for learning purposes it could still amount to an "I don't know".

Lastly, information about *ports* in the test cases could also be used. Considering a highly connectable SUT, such a system would feature a number of different ports connecting to different other systems. To learn a protocol automaton for just a subset of those communication ports, the test case traces could be restricted to the ports in question, excluding all others.

2) *Influence of Coverage:* While the semantic state-merging approach is able to make up for many missing traces, the case study suggests that the test suite used in learning must at least satisfy a path coverage of the SUT, as the one experiment where only a branch coverage was used failed. However, there are other coverage criteria besides branch and path coverage, e.g., based on condition determination or on the functions of the SUT, or automatic test case generation techniques. Further research is needed to clarify the dependencies of system structure, test suite coverage, and learnability.

C. Suitability of the Learning Algorithm

When confronted with the problem of reconstructing a system model from test cases, learning algorithms seemed to be a simple solution. The starting assumption was that while the test cases could be used as they were, some adaptations would have to be made to the algorithm. Instead, research shows that the learning algorithm itself can be used without changes, while the effort of adaptation concerns the representation of the learning sample, i.e., the test cases. In fact, the approach to learning from test cases proved to be a problem of teaching more than of learning.

1) *Online and Offline Learning:* Online learning algorithms, like Angluin's algorithm, build a system model by querying a teacher. Their main advantage is in generating the necessary queries themselves, thereby avoiding the need for a complete sample. However, this approach implies the existence of an omniscient oracle, which is able to answer arbitrary queries. In contrast, offline learning algorithms assume the existence of a structurally complete sample, which is then merged into the target automaton.

The query mechanisms used by Angluin's algorithm naturally match the test cases' verdicts. Also, Angluin's algorithm is scalable, growing only linearly with the size of the target automaton, and always generates a minimal DFA. As a test suite always assumes completeness with regard to certain coverage criteria, it can be assumed that the completeness of the test suite is sufficient to answer the membership queries. However, our experiments show that this assumption holds only for high coverage criteria and even then depends on the structure of the system.

These results seem to suggest that an offline learning approach would work better for the learning from test cases. Lambeau et al. [30] propose an offline algorithm based on state-merging, which takes into account merge constraints as well as incompatibility constraints, requiring some states to be merged obligatorily while others need to stay separated. This approach might work well for the learning from test cases. However, state-merging algorithms always need to be closely tailored to the sample data to merge. Therefore, a state-merging approach would only work for the special semantics it is designed for.

Our approach combines the advantages of both online and offline algorithms. The online algorithm is used to drive the overall learning process, thereby establishing a learning procedure that is independent from any given test specification language. Underlying the learning process, state-merging is used to mine the test language specific information for better coverage of the automaton's traces and to generate a data structure to be used as an oracle.

Following this layered approach, existing methods could be integrated for optimization. Regarding the online learning part, these optimizations concern the type of automaton generated, incorporating i.e., parameterization [27], [28]. Optimizations on the offline learning part should extend

the semantic state-merging approach. Possibly exploitable properties of test cases comprise differentiation of input and output signals and consideration of stable testing states. For example, the stable testing states could be matched to the merge constraints in the approach by Lambeau et al. [30].

2) *Other Versions of Angluin's Algorithm:* Another source for optimization of the learning procedure is the version of Angluin's algorithm that is used. The prototypical implementation uses a variation introduced by Kearns and Vazirani [29], which stores the gathered information in a classification tree. This version has the advantage of asking less membership queries, but at the cost of more equivalence queries [31]. Also, the classification tree provides a structure that is easy to maintain and therefore quickly to implement.

The original version of Angluin's algorithm uses an observation table to store the gathered information. This variation asks more membership queries before constructing the first hypothesis automaton, thereby reducing the number of needed equivalence queries [31]. On the other hand, maintaining the consistency of the observation table needs more effort.

Experimentation shows that using the trace graph as an oracle, membership queries are cheap, as their complexity only depends on the length of the queried trace. Equivalence queries take time, as in the worst case, the whole test suite has to be run against the hypothesis automaton. Also, the adaptations to learning from test cases are completely independent of the underlying implementation of Angluin's algorithm. Therefore, re-implementing the core of the learning algorithm according to the original version of Angluin's algorithm might even provide a small performance advantage.

3) *Breaking the Closed World Assumption:* In most learning scenarios, it is comparatively easy to get a correct answer to membership queries, while the equivalence query is hard to decide. When learning from a complete test suite, it is the other way around. The equivalence query can be matched easily to a run of the test suite against the hypothesis automaton, the only limiting factor being the time needed to run a large test suite. This also relates to the results of Berg et al. [32], who investigate the similarities between conformance testing and automata inference, finding that conformance testing solves a checking problem. Therefore, we can safely assume that a test suite that is sufficient to declare a system as conforming to its specification also suffices to decide whether it is equivalent to a learned hypothesis automaton.

In contrast, when asking membership queries against a limited set of traces, as every test suite is bound to be, there will always be queried traces that are not contained in the test suite. As the experiments show, rejecting every unknown trace can lead to bad generalization in the hypothesis automaton, while trying to provide for every possible query leads to inhibitive large test suites. There are several

possible approaches to solve this dilemma.

One way is to mine the test suite for implicit traces by state-merging. First efforts in this direction have been integrated into our learning approach and have shown positive results. Besides further exploitation of the properties of the test languages, also input from existing research in state-merging techniques can be used. The state-merging approach has two main advantages. The approach is self-contained, as no external input is needed, and it is safe, as the state-merging is based on information internal to the test suite. The drawback is that the mining depends on the information available in the test suite. If a test language with restricted description possibilities is used, the possibilities of state-merging are also restricted. Besides, while state-merging is able to boost the number of covered traces, it cannot make up for missing test cases if the test suites coverage is insufficient.

Another possible approach is to include explicit “don’t know” into the possible answers of a membership query. The problem of undecidable membership queries has occurred to researchers in other settings before, therefore a number of possibly adaptable methods exist. Sloan and Turán [33] define a meta-algorithm for incomplete membership oracles. For each undecidable membership query, the learning process is forked, one instance assuming acceptance, the other rejection of the queried string. If a copy is detected to be inconsistent, it is pruned. While this approach clearly leads to an exponential growth in the computation, the difficulty also is how to determine inconsistencies in the forked hypotheses. Bshouty and Owshanko [34] propose an extension of Angluin’s learning algorithm including “don’t know” as a possible answer to a membership query. Based on the Kearns-Vazirani version of the algorithm, they partition the possible traces of the target automaton into cover sets, resetting the algorithm when a counter-example causes a cover set to be split. Grinchtein and Leucker [35] also suggest an extension of Angluin’s algorithm. Using Angluin’s original version, they generate an incomplete observation table that they subsequently feed into a satisfiability solver, filling in the gaps with the most consistent solution. However, all those approaches share the disadvantage of replacing the uncertainties of the membership oracle with assumptions, thereby deviating from exact learning.

The third approach is a combination of passive and active techniques. In this approach, the algorithm learns as much as possible using the available information, fully exploiting every counter-example. When an unanswerable query is encountered, the query is either addressed at an external oracle, e.g., a domain expert, or translated into a test case that is executed against the SUT. Asking a domain expert leads to a guided learning approach. In this case, the learning is only semi-automatic. Executing a test case against an SUT could be conducted automatically. However, as the outcome of the query then would depend on the SUT, this approach

compromises the independence of the learned automaton. As both approaches draw information from sources beside the test suite, inconsistencies could be introduced into the learning data.

VIII. CONCLUSION

We have presented a learning approach that combines state-merging and learning techniques to generate a DFA from a test suite. The state-merging is used to represent the test suite and to find additional test cases exploiting the semantic properties of the test language. The combined approach has been implemented in a prototypical tool. Experiments show that while the state-merging approach reduces the size of the test suite needed for correct identification of the model, complex models still need a large number of test cases for correct identification.

We have discussed the design decisions that form the basis of our approach to learning from test. The main issue is the size and coverage of the test suite used in the learning process. While the mapping of test cases to learning traces is intuitive and simple, the size of a test suite sufficient for learning can get inhibitive large. Optimizations to deal with this problem comprise the extension of the semantic state-merging approach to better exploit the information contained in the test cases and an extension of the learning algorithm to work with unanswerable membership queries. In addition, the relation between test suite coverage, system structure, and learnability offers interesting research topics.

Based on the experiments with our learning approach, the next step is to incorporate the identified optimizations into our prototypical implementation. In the long run, our findings on the learnability of different models could also be used to assess the adequacy of a test suite.

REFERENCES

- [1] E. Werner and J. Grabowski, “Model Reconstruction: Mining Test Cases,” in *Third International Conference on Advances in System Testing and Validation Lifecycle (VALID 2011)*, Oct. 2011.
- [2] D. Angluin, “Learning Regular Sets from Queries and Counterexamples,” *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.
- [3] E. Werner, S. Polonski, and J. Grabowski, “Using Learning Techniques to Generate System Models for Online Testing,” in *Proc. INFORMATIK 2008*, ser. LNI, vol. 133. Köllen Verlag, 2008, pp. 183–186.
- [4] M. Shahbaz and R. Groz, “Inferring Mealy Machines,” in *Proc. FM 2009*, ser. LNCS, vol. 5850. Springer, 2009, pp. 207–222.
- [5] F. Aarts, B. Jonsson, and J. Uijen, “Generating Models of Infinite-State Communication Protocols Using Regular Inference with Abstraction,” in *Proc. ICTSS’10*, ser. LNCS, vol. 6435. Springer, 2010, pp. 188–204.

- [6] T. Berg, B. Jonsson, and H. Raffelt, "Regular Inference for State Machines Using Domains with Equality Tests," in *Proc. FASE 2008*, ser. LNCS, vol. 4961. Springer, 2008, pp. 317–331.
- [7] T. Bohlin, B. Jonsson, and S. Soleimanifard, "Inferring Compact Models of Communication Protocol Entities," in *proc. ISoLA 2010*, ser. LNCS, vol. 6415. Springer, 2010, pp. 658–672.
- [8] M. Shahbaz, K. Li, and R. Groz, "Learning and Integration of Parameterized Components Through Testing," in *Proc. TestCom 2007*, ser. LNCS, vol. 4581. Springer, 2007, pp. 319–334.
- [9] J. Esparza, M. Leucker, and M. Schlund, "Learning Workflow Petri Nets," in *Proc. PETRI NETS 2010*, ser. LNCS, vol. 6128. Springer, 2010, pp. 206–225.
- [10] B. Bollig, J.-P. Katoen, C. Kern, and M. Leucker, "SMA — The Smyle Modeling Approach," *Computing and Informatics*, vol. 29, no. 1, pp. 45–72, 2010.
- [11] A. W. Biermann and R. Krishnaswamy, "Constructing Programs from Example Computations," *IEEE Transactions on Software Engineering*, vol. 2, no. 3, pp. 141–153, 1976.
- [12] K. J. Lang, B. A. Pearlmutter, and R. A. Price, "Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm," in *Proc. ICGI-98*, ser. LNCS, vol. 1433. Springer, 1998, pp. 1–12.
- [13] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.
- [14] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," *TOSEM*, vol. 7, no. 3, pp. 215–249, 1998.
- [15] K. Koskimies and E. Mäkinen, "Automatic synthesis of state machines from trace diagrams," *Software—Practice & Experience*, vol. 24, no. 7, pp. 643–658, 1994.
- [16] I. H. Krüger and R. Mathew, "Component Synthesis from Service Specifications," in *Revised Selected Papers of the International Workshop on Scenarios: Models, Transformations and Tools*, ser. LNCS, vol. 3466. Springer, 2003, pp. 255–277.
- [17] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli, "Automatic Synthesis of Behavior Protocols for Composable Web-Services," in *Proc. ESEC/SIGSOFT FSE*. ACM, 2009, pp. 141–150.
- [18] C. Lee, F. Chen, and G. Rosu, "Mining parametric specifications," in *Proc. ICSE 2011*. ACM, 2011, pp. 591–600.
- [19] G. Ammons, R. Bodik, and J. R. Larus, "Mining Specifications," in *Proc. POPL'02*. ACM, 2002, pp. 4–16.
- [20] L. M. Duarte, J. Kramer, and S. Uchitel, "Model Extraction Using Context Information," in *Proc. MoDELS 2006*, ser. LNCS, vol. 4199. Springer, 2006, pp. 380–394.
- [21] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic Generation of Software Behavioral Models," in *Proc. ICSE 2008*. ACM, 2008, pp. 501–510.
- [22] *ETSI Standard (ES) 201 873: The Testing and Test Control Notation version 3; Parts 1–10*, ETSI Std., Rev. 4.2.1, 2010.
- [23] E. Werner, "Learning Finite State Machine Specifications from Test Cases," Ph.D. dissertation, Georg-August-Universität Göttingen, Göttingen, Jun. 2010. [Online]. Available: <http://webdoc.sub.gwdg.de/diss/2010/werner/>
- [24] L. D. Bousquet, S. Ramangalahy, S. Simon, C. Viho, A. F. E. Belinfante, and R. G. Vries, "Formal Test Automation: The Conference protocol with TGV/Torx," in *Proc. TestCom 2000*, ser. IFIP Conference Proceedings. Kluwer Academic Publishers, 2000, pp. 221–228.
- [25] D. Neumann, "Test Case Generation using Model Transformations," Master's Thesis, University of Göttingen, Institute for Computer Science, Göttingen, Germany, 2009.
- [26] T. Berg, B. Jonsson, M. Leucker, and M. Saksena, "Insights to Angluin's Learning," *ENTCS*, vol. 118, pp. 3–18, 2005.
- [27] T. Berg, B. Jonsson, and H. Raffelt, "Regular Inference for State Machines with Parameters," in *Proc. FASE 2006*, ser. LNCS, vol. 3922. Springer, 2006, pp. 107–121.
- [28] K. Li, R. Groz, and M. Shahbaz, "Integration Testing of Distributed Components Based on Learning Parameterized I/O Models," in *Proc. FORTE 2006*, ser. LNCS, vol. 4229. Springer, 2006, pp. 436–450.
- [29] M. J. Kearns and U. V. Vazirani, *An Introduction to Computational Learning Theory*. MIT Press, 1994.
- [30] B. Lambeau, C. Damas, and P. Dupont, "State-Merging DFA Induction Algorithms with Mandatory Merge Constraints," in *Proc. ICGI 2008*, ser. LNCS, vol. 5278. Springer, 2008, pp. 139–153.
- [31] J. L. Balcázar, J. Díaz, R. Gavaldà, and O. Watanabe, "Algorithms for Learning Finite Automata from Queries: A Unified View," in *Advances in Algorithms, Languages, and Complexity*, D.-Z. Du and K.-I. Ko, Eds. Kluwer Academic Publishers, 1997, pp. 53–72.
- [32] T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, and B. Steffen, "On the Correspondence Between Conformance Testing and Regular Inference," in *Proc. FASE 2005*, ser. LNCS, vol. 3442. Springer, 2005, pp. 175–189.
- [33] R. H. Sloan and G. Turán, "Learning with queries but incomplete information (extended abstract)," in *Proc. COLT '94*. ACM, 1994, pp. 237–245.
- [34] N. H. Bshouty and A. Owshanko, "Learning Regular Sets with an Incomplete Membership Oracle," in *Proc. COLT 2001 and EuroCOLT 2001*, ser. LNCS, vol. 2111. Springer, 2001, pp. 574–588.
- [35] O. Grinchtein and M. Leucker, "Learning Finite-State Machines from Inexperienced Teachers," in *Proc. ICGI 2006*, ser. LNCS, vol. 4201. Springer, 2006, pp. 344–345.

Synthesizing Control Software from Boolean Relations

Federico Mari, Igor Melatti, Ivano Salvo, and Enrico Tronci

Department of Computer Science

Sapienza University of Rome

Via Salaria 113, 00198 Rome, Italy

Email: {mari,melatti,salvo,tronci}@di.uniroma1.it

Abstract—Many software as well digital hardware automatic synthesis methods define the set of implementations meeting the given system specifications with a boolean relation K . In such a context a fundamental step in the software (hardware) synthesis process is finding effective solutions to the functional equation defined by K . This entails finding a (set of) boolean function(s) F (typically represented using OBDDs, *Ordered Binary Decision Diagrams*) such that: 1) for all x for which K is satisfiable, $K(x, F(x)) = 1$ holds; 2) the implementation of F is efficient with respect to given implementation parameters such as code size or execution time. While this problem has been widely studied in digital hardware synthesis, little has been done in a software synthesis context. Unfortunately, the approaches developed for hardware synthesis cannot be directly used in a software context. This motivates investigation of effective methods to solve the above problem when F has to be implemented with software. In this paper, we present an algorithm that, from an OBDD representation for K , generates a C code implementation for F that has the same size as the OBDD for F and a worst case execution time linear in nr , being $n = |x|$ the number of input arguments for functions in F and r the number of functions in F . Moreover, a formal proof of the proposed algorithm correctness is also shown. Finally, we present experimental results showing effectiveness of the proposed algorithm.

Keywords-Control Software Synthesis; Embedded Systems; Model Checking

I. INTRODUCTION

Many software as well digital hardware automatic synthesis methods define the set of implementations meeting the given system specifications with a boolean relation K . Given an n -bits (resp., r -bits) *binary encoding* of states (resp., actions) of the system as it is usually done in Model Checking [7] (see Sect. III-B), such relation typically takes as input the n -bits encoding of a state x and the r -bits encoding of a proposed action to be performed u , and returns *true* (i.e., 1) if and only if the system specifications are met when performing action u in state x . In such a context, a fundamental step in the software (hardware) synthesis process is finding effective solutions to the functional equation defined by K , i.e., $K(x, u) = 1$. This entails finding a tuple of boolean functions $F = \langle f_1, \dots, f_r \rangle$ (typically represented using OBDDs, *Ordered Binary Decision Diagrams* [2]) such that 1) for all x for which K is satisfiable (i.e., it enables at least one action), $K(x, F(x)) = 1$ holds, and 2) the implementation of F is efficient with respect to given

implementation parameters such as code size or execution time.

While this problem has been widely studied in digital hardware synthesis [3][4], little has been done in a software synthesis context. This is not surprising since software synthesis from formal specifications is still in its infancy. Unfortunately the approaches developed for hardware synthesis cannot be directly used in a software context. In fact, synthesis methods targeting a hardware implementation typically aim at minimizing the number of digital gates and of hierarchy levels. Since in the same hierarchy level gates output computation is *parallel*, the hardware implementation WCET (*Worst Case Execution Time*) is given by the number of levels. On the other hand, a software implementation will have to *sequentially* compute the gates outputs. This implies that the software implementation WCET is the number of gates used, while a synthesis method targeting a software implementation may obtain a better WCET. This motivates investigation of effective methods to solve the above problem when F has to be implemented with software.

A. Our Contribution

In this paper, we present an algorithm that, from an OBDD representation for K , effectively generates a C code implementation for K that has the same size as the OBDD for F and a WCET linear in linear in nr , being $n = |x|$ the number of bits encoding state x and $r = |u|$ the number of bits encoding action u . This is done in two steps:

- 1) from an OBDD representation for K we effectively compute an OBDD representation for F , following the lines of [5];
- 2) we generate a C code implementation for F with the above described properties of code size and WCET.

We formally prove both steps 1 and 2 to be correct.

This allows us to synthesize correct-by-construction *control software*, provided that K is provably correct with respect to initial formal specifications. This is the case of [6], where an algorithm is presented to synthesize K starting from a) the formal specification of a Discrete-Time Linear Hybrid System (*DTLHS* in the following) modeling the system (plant) to be controlled, b) its system level formal specifications (specifying the goal to be reached and the safe states to be traversed in order to reach it)

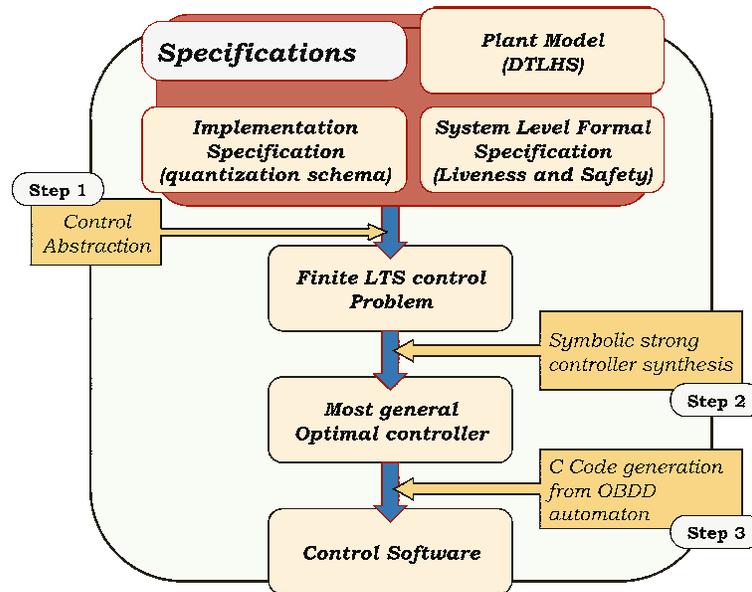


Figure 1. Control Software Synthesis Flow

and c) the quantization schema (i.e., the number of bits available for analog-to-digital conversion). The framework in [6] is depicted in Figure 1. With respect to Figure 1, the approach proposed in this paper may be used to perform step 3. Thus, this methodology allows a correct-by-construction control software to be synthesized, starting from formal specifications for DTLHSs.

Note that the problem of solving the functional equation $K(x, F(x)) = 1$ with respect to F is trivially decidable, since there are finitely many F . However, trying to explicitly enumerate all F requires time $\Omega(2^{r2^n})$. By using OBDD-based computations, we are able to compute F in time $O(r2^n)$ in the worst case. However, in many interesting cases OBDD sizes and computations are much lower than the theoretical worst case (e.g., in Model Checking applications, see [7]).

Furthermore, once the OBDD representation for F has been computed, a trivial implementation of F could use a look-up table in RAM. While this solution would yield a better WCET, it would imply a $\Omega(r2^n)$ RAM usage. Unfortunately, implementations for F in real-world cases are typically implemented on microcontrollers (this is the case, e.g., for *embedded systems*). Since microcontrollers usually have a small RAM, the look-up table based solution is not feasible in many interesting cases. The approach we present here will rely on OBDDs compression to overcome such obstruction.

Moreover, $F : \mathbb{B}^n \rightarrow \mathbb{B}^r$ is composed by r boolean functions, thus it is represented by r OBDDs. Such OBDDs typically share nodes among them. If a trivial implementation of F in C code is used, i.e., each OBDD is translated as a stand-alone C function, such inter-OBDDs nodes sharing

will not be exploited. In our approach, we exploit inter-OBDDs nodes sharing, thus the control software we generate fully takes advantage of OBDDs compression.

Finally, we present experimental results showing effectiveness of the proposed algorithm. As an example, in less than 1 second and within 350 MB of RAM we are able to synthesize the control software for a function K of 25 boolean variables, divided in $n = 20$ state variables and $r = 5$ action variables, represented by an OBDD with about 6.6×10^4 nodes. Such K represents the set of correct implementations for a real-world system, namely a multi-input buck DC/DC converter [8], obtained as described in [6]. The control software we synthesize in such a case has about 1.7×10^4 lines of code, whilst a control software not taking into account OBDDs nodes sharing would have had about 2.1×10^4 lines of code. Thus, we obtain a 20% gain towards a trivial implementation.

This paper is organized as follows. In Section III we give the basic notions to understand our approach. In Section IV we formally define the problem we want to solve. In Section V we give definition and main properties of COBDDs (i.e., *Complemented edges OBDDs*), on which our approach is based. Section VI describes the algorithms our approach consists of, whilst Section VII proves it to be correct. Section VIII presents experimental results showing effectiveness of the proposed approach. Finally, Section IX presents the concluding remarks and gives some ideas for future work.

II. RELATED WORK

This paper is an extended version of [1]. With respect to [1], this paper provides more details in the introduction

and in the related work description, extends basic definitions and algorithms descriptions, shows omitted proofs for theorems and provides a revised version of the experiments.

Synthesis of boolean functions F satisfying a given boolean relation K in a way such that $K(x, F(x)) = 1$ is also addressed in [3]. However, [3] targets a hardware setting, whereas we are interested in a software implementation for F . Due to structural differences between hardware and software based implementations (see the discussion in Section I), the method in [3] is not directly applicable here. An OBDD-based method for synthesis of boolean (reversible) functions is presented in [4] (see also citations thereof). Again, the method in [4] targets a hardware implementation, thus it is not applicable here.

An algorithm for the synthesis of C control software is also presented in [9]. However, in [9] the starting point is a (multioutput) boolean function, rather than a boolean relation. That is to say, the starting point is F rather than K (with respect to the discussion in Section I-A, it is supposed that step 1 has already been performed). Moreover, the algorithm in [9], though OBDD-based, does not generate a software with the same size of the OBDDs for F , nor an estimation of its WCET (in the sense explained in Section I) is provided. Finally, an implementation of the algorithm in [9] is not provided, thus we cannot make a direct experimental comparison with our method.

Synthesis of control software is also addressed in [10], where the focus is on the generation of control protocols. Such method cannot be applied in our context, where we need a C software implementation.

In [6], an algorithm is presented which, starting from a formal specification of a DTLHS, synthesizes a correct-by-construction boolean relation K , and then a correct-by-construction control software implementation for K (see Figure 1). However, in [6] the implementation of K is not described in detail. Furthermore, the implementation synthesis described in [6] has not the same size of the OBDD for F , i.e., it does not exploit OBDD nodes sharing.

Many other works in the literature has the goal of synthesizing controllers as boolean relations K , under very different assumptions for the target dynamic system to be controlled. Such works do not deal with the effective implementation of K , thus they may use the approach described here in order to have an effective software implementation of K . As an example, the following works may be cited as closer to ours. In [11] controllers are generated starting from finite-state nondeterministic dynamic systems (arising from planning problems). In [12] a method to synthesize non-optimal (but smaller in size) controllers is presented.

In [5], an algorithm is presented which computes boolean functions F satisfying a given boolean relation K in a way such that $K(x, F(x)) = 1$. This approach is very similar to ours. However [5] does not generate the C code control software and it does not exploit OBDD nodes sharing.

Finally, we note that our work lies in the wider area of software synthesis, which has been widely studied since a long time in many contexts. For a survey on such (non-control) software synthesis works, see [13][14].

III. BASIC DEFINITIONS

In the following, we denote with $\mathbb{B} = \{0, 1\}$ the boolean domain, where 0 stands for *false* and 1 for *true*. We will denote boolean functions $f : \mathbb{B}^n \rightarrow \mathbb{B}$ with boolean expressions on boolean variables involving $+$ (logical OR), \cdot (logical AND, usually omitted thus $xy = x \cdot y$), $\bar{}$ (logical complementation) and \oplus (logical XOR). We will also denote vectors of boolean variables in boldface, e.g., $\mathbf{x} = \langle x_1, \dots, x_n \rangle$. Moreover, we also denote with $f|_{x_i=g}(\mathbf{x})$ the boolean function $f(x_1, \dots, x_{i-1}, g(\mathbf{x}), x_{i+1}, \dots, x_n)$ and with $\exists x_i f(\mathbf{x})$ the boolean function $f|_{x_i=0}(\mathbf{x}) + f|_{x_i=1}(\mathbf{x})$.

Finally, we denote with $[n]$ the set $\{1, \dots, n\}$.

A. Most General Optimal Controllers

A *Labeled Transition System* (LTS) is a tuple $\mathcal{S} = (S, A, T)$ where S is a finite set of *states*, A is a finite set of *actions*, and T is the (possibly non-deterministic) *transition relation* of \mathcal{S} . A *controller* for an LTS \mathcal{S} is a function $K : S \times A \rightarrow \mathbb{B}$ enabling actions in a given state. We denote with $\text{Dom}(K)$ the set of states for which a control action is enabled. An LTS *control problem* is a triple $\mathcal{P} = (\mathcal{S}, I, G)$, where \mathcal{S} is an LTS and $I, G \subseteq S$. A controller K for \mathcal{S} is a *strong solution* to \mathcal{P} if and only if it drives each *initial state* $s \in I$ in a *goal state* $t \in G$, notwithstanding nondeterminism of \mathcal{S} . A strong solution K^* to \mathcal{P} is *optimal* if and only if it minimizes path lengths. An optimal strong solution K^* to \mathcal{P} is the *most general optimal controller* (we call such solution an *mgo*) if and only if in each state it enables all actions enabled by other optimal controllers. For more formal definitions of such concepts, see [15].

Efficient algorithms, typically reminiscent of early work on minimum paths by Dijkstra [16], to compute controllers starting from suitable (nondeterministic) LTS control problems have been proposed in the literature: e.g., [11] presents an algorithm to generate mgos, while [12] show an algorithm for non-optimal (but smaller in size) controllers. Once a controller K has been computed, solving and implementing the functional equation $K(\mathbf{x}, \mathbf{u}) = 1$ allows a correct-by-construction control software to be synthesized.

B. Binary Encoding for States and Actions

Vectors of boolean values $\mathbf{x} \in \mathbb{B}^n$ (resp., $\mathbf{u} \in \mathbb{B}^r$) may be used to represent states $s \in S$ (resp., actions $a \in A$) of an LTS $\mathcal{S} = (S, A, T)$ (and thus of a controller for \mathcal{S}) as follows. Let $n = \lfloor \log_2(|S|) \rfloor + 1$. Then, n boolean values (bits) may be used to represent any $s \in S$. As an example, in Model Checking applications [7] an order on $S = \{s_1, \dots, s_m\}$ is fixed (let $s_1 < \dots < s_m$ be such order), and then the binary encoding $\eta : S \rightarrow \mathbb{B}^n$ is defined

as $\eta(s_i) = \mathbf{b}$ such that $\sum_{j=1}^n 2^{j-1} b_j = i - 1$. An analogous construction may be applied to actions.

C. OBDD Representation for Boolean Functions

A *Binary Decision Diagram* (BDD) R is a rooted directed acyclic graph (DAG) with the following properties. Each R node v is labeled either with a boolean variable $\text{var}(v)$ (internal node) or with a boolean constant $\text{val}(v) \in \mathbb{B}$ (terminal node). Each R internal node v has exactly two children, labeled with $\text{high}(v)$ and $\text{low}(v)$. Let x_1, \dots, x_n be the boolean variables labeling R internal nodes. Each terminal node v represents $f_v(\mathbf{x}) = \text{val}(v)$. Each internal node v represents $f_v(\mathbf{x}) = x_i f_{\text{high}(v)}(\mathbf{x}) + \bar{x}_i f_{\text{low}(v)}(\mathbf{x})$, being $x_i = \text{var}(v)$. An *Ordered BDD* (OBDD) is a BDD where, on each path from the root to a terminal node, the variables labeling each internal node must follow the same ordering.

IV. SOLVING A BOOLEAN FUNCTIONAL EQUATION

Let $K(x_1, \dots, x_n, u_1, \dots, u_r)$ be the mgo for a given control problem $\mathcal{P} = (\mathcal{S}, I, G)$. We want to solve the *boolean functional equation* $K(\mathbf{x}, \mathbf{u}) = 1$ with respect to variables \mathbf{u} , that is we want to obtain boolean functions f_1, \dots, f_r such that $K(\mathbf{x}, f_1(\mathbf{x}), \dots, f_r(\mathbf{x})) = K|_{u_1=f_1(\mathbf{x}), \dots, u_r=f_r(\mathbf{x})}(\mathbf{x}, \mathbf{u}) = 1$. This problem may be solved in different ways, depending on the *target implementation* (hardware or software) for functions f_i . In both cases, it is crucial to be able to bound the WCET (*Worst Case Execution Time*) of the obtained controller. In fact, controllers must work in an endless closed loop with the system \mathcal{S} (*plant*) they control. This implies that, every T seconds (*sampling time*), the controller has to determine the actions to be sent to \mathcal{S} . Thus, in order for the entire system (plant + control software) to properly work, the controller WCET upper bound must be at most T .

In [3], f_1, \dots, f_r are generated in order to optimize a *hardware* implementation. In this paper, we focus on software implementations for f_i (*control software*). As it is discussed in Section I, simply translating an hardware implementation into a software implementation would result in a too high WCET. Thus, a method directly targeting software is needed. An easy solution would be to set up, for a given state \mathbf{x} , a SAT problem instance $\mathcal{C} = C_{K1}, \dots, C_{Kt}, c_1, \dots, c_n$, where $C_{K1} \wedge \dots \wedge C_{Kt}$ is equisatisfiable to K and each clause c_i is either x_i (if x_i is 1) or \bar{x}_i (otherwise). Then \mathcal{C} may be solved using a SAT solver, and the values assigned to \mathbf{u} in the computed satisfying assignment may be returned as the action to be taken. However, it would be hard to estimate a WCET for such an implementation. The method we propose in this paper overcomes such obstructions by achieving a WCET proportional to rn .

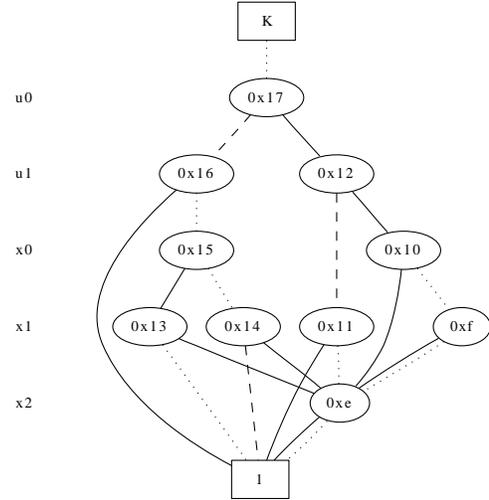


Figure 2. An mgo example

V. OBDDs WITH COMPLEMENTED EDGES

In this section, we introduce OBDDs with complemented edges (COBDDs, Definition 1), which were first presented in [17][18]. Intuitively, they are OBDDs where else edges (i.e., edges of type $(v, \text{low}(v))$) may be complemented. Then edges (i.e., edges of type $(v, \text{high}(v))$) complementation is not allowed to retain canonicity. Edge complementation usually reduce resources usage, both in terms of CPU and memory.

Definition 1. An *OBDD with complemented edges* (COBDD in the following) is a tuple $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ with the following properties:

- 1) $\mathcal{V} = \{x_1, \dots, x_n\}$ is a finite *ordered* set of boolean variables;
- 2) V is a finite set of *nodes*;
- 3) $\mathbf{1} \in V$ is the *terminal* node of ρ , corresponding to the boolean constant 1 (non-terminal nodes are called *internal*);
- 4) $\text{var}, \text{low}, \text{high}, \text{flip}$ are functions defined on internal nodes, namely:
 - $\text{var} : V \setminus \{\mathbf{1}\} \rightarrow \mathcal{V}$ assigns to each internal node a boolean variable in \mathcal{V}
 - $\text{high}[\text{low}] : V \setminus \{\mathbf{1}\} \rightarrow V$ assigns to each internal node v a *high child* [*low child*] (or *then child* [*else child*]), representing the case in which $\text{var}(v) = 1$ [$\text{var}(v) = 0$]
 - $\text{flip} : V \setminus \{\mathbf{1}\} \rightarrow \mathbb{B}$ assigns to each internal node v a boolean value; namely, if $\text{flip}(v) = 1$ then the else child has to be complemented, otherwise it is regular (i.e., non-complemented);
- 5) for each internal node v , $\text{var}(v) < \text{var}(\text{high}(v))$ and $\text{var}(v) < \text{var}(\text{low}(v))$.

A. COBDDs Associated Multigraphs

We associate to a COBDD $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ a labeled directed multigraph $G^{(\rho)} = (V, E)$ such that V is the same set of nodes of ρ and there is an edge $(v, w) \in E$ if and only if w is a child of v . Moreover, each edge $e = (v, w) \in E$ has a type $\text{type}(e)$, indicating if e is a *then edge* (i.e., if w is a then child of v), a *regular else edge* (i.e., if w is an else child of v with $\text{flip}(v) = 0$), or a *complemented else edge* (i.e., if w is an else child of v with $\text{flip}(v) = 1$). Figure 2 shows an example of a COBDD depicted via its associated multigraph, where edges are directed downwards. Moreover, in Figure 2 then edges are solid lines, regular else edges are dashed lines and complemented else edges are dotted lines.

The graph associated to a given COBDD $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ may be seen as a forest with multiple rooted multigraphs. In order to select one root vertex and thus one rooted multigraph, we define the *COBDD restricted to $v \in V$* as the COBDD $\rho_v = (\mathcal{V}, V_v, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ such that $V_v = \{w \in V \mid \text{there exists a path from } v \text{ to } w \text{ in } G^{(\rho)}\}$ (note that $v \in V_v$).

B. COBDDs Properties

For a given COBDD $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ the following properties follow from definitions given above:

- 1) $G^{(\rho)}$ is a rooted directed acyclic (multi)graph (DAG);
- 2) each path in $G^{(\rho)}$ starting from an internal node ends in $\mathbf{1}$;
- 3) let v_1, \dots, v_k be a path in $G^{(\rho)}$, then $\text{var}(v_1) < \dots < \text{var}(v_k)$.

We define the *height of a node v in a COBDD ρ* (notation $\text{height}_\rho(v)$, or simply $\text{height}(v)$ if ρ is understood) as the height of the DAG $G^{(\rho_v)}$, i.e., the length of the longest path from v to $\mathbf{1}$ in $G^{(\rho)}$.

C. Semantics of a COBDD

In Definition 2, we define the semantics $\llbracket \cdot \rrbracket$ of each node v of a given COBDD ρ as the boolean function represented by v , given the parity b of complemented edges seen on the path from a root to v .

Definition 2. Let $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD. The *semantics of the terminal node $\mathbf{1}$ with respect to a flipping bit b* is a boolean function defined as $\llbracket \mathbf{1}, b \rrbracket_\rho := \bar{b}$. The *semantics of an internal node $v \in V$ with respect to a flipping bit b* is a boolean function defined as $\llbracket v, b \rrbracket_\rho := x_i \llbracket \text{high}(v), b \rrbracket_\rho + \bar{x}_i \llbracket \text{low}(v), b \oplus \text{flip}(v) \rrbracket_\rho$, being $x_i = \text{var}(v)$. When ρ is understood, we will write $\llbracket \cdot \rrbracket$ instead of $\llbracket \cdot \rrbracket_\rho$.

Note that the semantics of a node of a COBDD ρ is a function of variables in \mathcal{V} and of an additional boolean variable b . Thus, on each node *two* boolean functions on \mathcal{V} are defined (one for each value of b). It can be shown (see [15]) that such boolean functions are complementary.

Example 1. Let ρ be the COBDD depicted in Figure 2. If we pick node $0xe$ we have $\llbracket 0xe, b \rrbracket = x_2 \llbracket \mathbf{1}, b \rrbracket + \bar{x}_2 \llbracket \mathbf{1}, b \oplus \mathbf{1} \rrbracket = x_2 \bar{b} + \bar{x}_2 b = x_2 \oplus b$.

D. Reduced COBDDs and COBDDs Canonicity

Two COBDDs are *isomorphic* if and only if there exists a mapping from nodes to nodes preserving attributes var , flip , high and low . A COBDD is called *reduced* if and only if it contains no vertex v with $\text{low}(v) = \text{high}(v) \wedge \text{flip}(v) = 0$, nor does it contains distinct vertices v and v' such that ρ_v and $\rho_{v'}$ are isomorphic. Note that, differently from OBDDs, it is possible that $\text{high}(v) = \text{low}(v)$ for some $v \in V$, provided that $\text{flip}(v) = 1$ (e.g., see nodes $0xf$ and $0xe$ in Figure 2).

Theorem 1 states that reduced COBDDs are a *canonical* representation for boolean functions (see [17][18]). As a consequence, software packages implementing COBDDs operations only deal with reduced COBDDs, since this allows very fast equality tests between COBDDs (it is sufficient to check if the (root node, flipping bit) pair is the same). Accordingly, in the following we will deal with reduced COBDDs only.

Theorem 1. Let $f : \mathbb{B}^n \rightarrow \mathbb{B}$ be a boolean function. Then there exists a reduced COBDD $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$, a node $v \in V$ and a flipping bit $b \in \mathbb{B}$ such that $\llbracket v, b \rrbracket = f(x)$. Moreover, let $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a reduced COBDD, let $v_1, v_2 \in V$ be nodes and $b_1, b_2 \in \mathbb{B}$ be flipping bits. Then $\llbracket v_1, b_1 \rrbracket = \llbracket v_2, b_2 \rrbracket$ if and only if $v_1 = v_2 \wedge b_1 = b_2$.

VI. SYNTHESIS OF C CODE FROM A COBDD

Let $K(x_1, \dots, x_n, u_1, \dots, u_r)$ be a controller for a given control problem. Let $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD such that there exist $v \in V$, $b \in \mathbb{B}$ such that $\llbracket v, b \rrbracket = K(x_1, \dots, x_n, u_1, \dots, u_r)$. Thus, $\mathcal{V} = \mathcal{X} \cup \mathcal{U} = \{x_1, \dots, x_n\} \cup \{u_1, \dots, u_r\}$ (we denote with \cup the disjoint union operator, thus $\mathcal{X} \cap \mathcal{U} = \emptyset$). We will call variables $x_i \in \mathcal{X}$ as *state variables* and variables $u_j \in \mathcal{U}$ as *action variables*.

We want to solve the boolean functional equation problem introduced in Sect. IV targeting a *software* implementation. We do this by using a COBDD representing all our boolean functions. This allows us to exploit COBDD nodes sharing. This results in an improvement for the method in [5], which targets a software implementation but which does not exploit sharing. Finally, we also synthesize the software (i.e., C code) implementation for f_1, \dots, f_r , which is not considered in [5]. This allows us to finally have a *control software* for the starting LTS. If K is an mgo, this results in an *optimal control software* for the starting LTS.

A. Synthesis Algorithm: Overview

Our method *Synthesize* takes as input ρ , v and b such that $\llbracket v, b \rrbracket = K(x, u)$. Then, it returns as output a C function `void K(int *x, int *u)` with the following

property: if, before a call to K , $\forall i \ x[i-1] = x_i$ holds (array indexes in C language begin from 0) with $\mathbf{x} \in \text{Dom}(K)$, and after the call to K , $\forall i \ u[i-1] = u_i$ holds, then $K(\mathbf{x}, \mathbf{u}) = 1$. Moreover, the WCET of function K is $O(nr)$.

Note that our method *Synthesize* provides an effective implementation of the controller K , i.e., a C function which takes as input the current state of the LTS and outputs the action to be taken. Thus, K is indeed a control software.

Function *Synthesize* is organized in two phases. First, starting from ρ , v and b (thus from $K(\mathbf{x}, \mathbf{u})$), we generate COBDD nodes v_1, \dots, v_r and flipping bits b_1, \dots, b_r for boolean functions f_1, \dots, f_r such that each $f_i = \llbracket v_i, b_i \rrbracket$ takes as input the state bit vector \mathbf{x} and computes the i -th bit u_i of an output action bit vector \mathbf{u} , where $K(\mathbf{x}, \mathbf{u}) = 1$, provided that $\mathbf{x} \in \text{Dom}(K)$. This computation is carried out in function *SolveFunctionalEq*. Second, f_1, \dots, f_r are translated inside function `void K(int *x, int *u)`. This step is performed by maintaining the structure of the COBDD nodes representing f_1, \dots, f_r . This allows us to exploit COBDD nodes sharing in the generated software. This phase is performed by function *GenerateCCCode*.

Thus, function *Synthesize* is organized as in Algorithm 1. Correctness for function *Synthesize* is stated in Theorem 6.

Algorithm 1 Translating COBDDs to a C function

Require: COBDD ρ , node v , boolean b

Ensure: *Synthesize*(ρ, v, b):

- 1: $\langle v_1, b_1, \dots, v_r, b_r \rangle \leftarrow \text{SolveFunctionalEq}(\rho, v, b)$
 - 2: *GenerateCCCode*($\rho, v_1, b_1, \dots, v_r, b_r$)
-

B. Synthesis Algorithm: Solving a Functional Equation

In this phase, starting from ρ , v and b (thus from $\llbracket v, b \rrbracket = K(\mathbf{x}, \mathbf{u})$), we compute functions f_1, \dots, f_r such that for all $\mathbf{x} \in \text{Dom}(K)$, $K(\mathbf{x}, f_1(\mathbf{x}), \dots, f_r(\mathbf{x})) = 1$.

To this aim, we follow an approach similar to the one presented in [5], which is reminiscent of early work on minimum paths by Dijkstra. Namely, we compute f_i using f_1, \dots, f_{i-1} , in the following way: $f_i(\mathbf{x}) = \exists u_{i+1}, \dots, u_n \ K(\mathbf{x}, f_1(\mathbf{x}), \dots, f_{i-1}(\mathbf{x}), 1, u_{i+1}, \dots, u_n)$. Thus, function *SolveFunctionalEq*(ρ, v, b) computes and returns $\langle v_1, b_1, \dots, v_r, b_r \rangle$ such that for all $i \in [r]$, $\llbracket v_i, b_i \rrbracket = f_i(\mathbf{x})$. This is effectively performed by Algorithm 2, where we use the following COBDDs manipulation functions:

- *COBDD_APP* (instantiation) such that $\langle v_{APP}, b_{APP} \rangle = \text{COBDD_APP}(x_{i_1}, \dots, x_{i_k}, v_1, b_1, \dots, v_k, b_k, v, b)$ if and only if $\llbracket v_{APP}, b_{APP} \rrbracket = \llbracket v, b \rrbracket|_{x_{i_1}=\llbracket v_1, b_1 \rrbracket, \dots, x_{i_k}=\llbracket v_k, b_k \rrbracket}$;
- *COBDD_EX* (existential quantifier elimination) such that $\langle v_{EX}, b_{EX} \rangle = \text{COBDD_EX}(x_{i_1}, \dots, x_{i_k}, v, b)$ if and only if $\llbracket v_{EX}, b_{EX} \rrbracket = \exists x_{i_1}, \dots, x_{i_k} \llbracket v, b \rrbracket$.

We note that efficient (i.e., at most $O(|V| \log |V|)$) algorithms [17][18] exist to compute the above defined functions. Moreover, the above defined functions may create new

COBDD nodes. We assume that such functions also properly update V , var , low , high , flip inside COBDD ρ (1 and \mathcal{V} are not affected).

Algorithm 2 Solving a boolean functional equation

Require: COBDD ρ , node v , boolean b

Ensure: *SolveFunctionalEq*(ρ, v, b):

- 1: **for all** $i \in [r]$ **do**
 - 2: $\llbracket v_i, b_i \rrbracket \leftarrow \text{COBDD_EX}(u_{i+1}, \dots, u_n, \text{COBDD_APP}(u_1, \dots, u_i, v_1, b_1, \dots, v_{i-1}, b_{i-1}, 1, 0, v, b))$
 - 3: **return** $\langle v_1, b_1, \dots, v_r, b_r \rangle$
-

Correctness for function *SolveFunctionalEq* is proved in Lemma 3.

C. Synthesis Algorithm: Generating C Code

In this phase, starting from COBDD nodes v_1, \dots, v_r and flipping bits b_1, \dots, b_r for functions f_1, \dots, f_r generated in the first phase, we generate two C functions: i) `void K(int *x, int *u)`, which is the required output function for our method *Synthesize*; ii) `int K_bits(int *x, int action)`, which is an auxiliary function called by K . A call to `K_bits(x, i)` returns $f_i(\mathbf{x})$, being $x[j-1] = x_j$ for all $j \in [n]$. This phase is detailed in Algs. 3 (function *GenerateCCCode*) and 4 (function *Translate*). In such algorithms we suppose to be able to print a node v , e.g., by printing the hexadecimal value of a pointer to v .

Algorithm 3 Generating C functions

Require: COBDD ρ , v_1, \dots, v_r , boolean values b_1, \dots, b_r

Ensure: *GenerateCCCode*($\rho, v_1, b_1, \dots, v_r, b_r$):

- 1: **print** “int K_bits(int *x, int action) {
int ret_b; switch(action) {”
 - 2: **for all** $i \in [r]$ **do**
 - 3: **print** “case ”, $i-1$, “:”
 - 4: **print** “ret_b = ”, \bar{b}_i , “; goto L_”, v_i , “;”
 - 5: **print** “}” /* end of the switch block */
 - 6: $W \leftarrow \emptyset$
 - 7: **for all** $i \in [r]$ **do**
 - 8: $W \leftarrow \text{Translate}(\rho, v_i, W)$
 - 9: **print** “} K(int* x, int* u) {int i;”
 - 10: **print** “ for(i=0; i<”, r , “; i++)”
 - 11: **print** “ u[i] = K_bits(x, i);}”
-

Details of Function GenerateCCCode (Algorithm 3):
Given inputs ρ , $v_1, b_1, \dots, v_r, b_r$ (output by *SolveFunctionalEq*), Algorithm 3 works as follows. First, function `int K_bits(int *x, int action)` is generated. If $x[j-1] = x_j$ for all $j \in [n]$, the call `K_bits(x, i)` has to return $f_i(\mathbf{x})$. In order to do this, the graph $G^{(\rho_{v_i})}$ is traversed by taking, in each node v , the then edge if $x[j-1] = 1$ (with j such that $\text{var}(v) = x_j$) and the else edge

otherwise. When node 1 is reached, then 1 is returned if and only if the integer sum $c + b_i$ is even, being c the number of complemented else edges traversed. Note that parity of $c + b_i$ may be maintained by initializing a C variable `ret_b` to \bar{b}_i , then complementing `ret_b` (i.e., by performing a `ret_b = !ret_b` statement) when a complemented else edge is traversed, and finally returning `ret_b`.

This mechanism is implemented inside function `K_bits` by properly translating each COBDD node $\tilde{v} \in \bigcup_{i=1}^r V_{v_i}$ in a C code block. Each block is labeled with a unique label depending on \tilde{v} , and maintains in variable `ret_b` the current parity of $c + b_i$ as described above. This is done by function *Translate*, called on line 8 and detailed in Algorithm 4.

Thus, the initial part of function `K_bits` consists of a switch block (generated in lines 1–5 of Algorithm 3), which initializes `ret_b` to \bar{b}_i and then jumps to the label corresponding to node v_i . Then, the C code blocks corresponding to COBDD nodes are generated in lines 6–8 of Algorithm 3, by calling r times function *Translate* (see Algorithm 4) with parameters v_1, \dots, v_r . Note that W maintains the already translated COBDD nodes. Since function *Translate* only translates nodes not in W , this allows us to exploit sharing not only inside each $G^{(\rho_{v_i})}$, but also inside $G^{(\rho_{v_1})}, \dots, G^{(\rho_{v_r})}$.

Finally, function `K` is generated in lines 9–11. Function `K` simply consists in a `for` loop filling each entry `u[i]` of the output array `u` with the boolean values returned by `K_bits(x, i)`. Correctness of function *GenerateCCode* is proved in Lemma 5.

Algorithm 4 COBDD nodes translation

Require: COBDD ρ , node v , nodes set W

Ensure: *Translate*(ρ, v, W):

```

1: if  $v \in W$  then return  $W$ 
2:  $W \leftarrow W \cup \{v\}$ 
3: print “L_”,  $v$ , “:”
4: if  $v = 1$  then
5:   print “return ret_b;”
6: else
7:   let  $i$  be such that  $\text{var}(v) = x_i$ 
8:   print “if(x[”,  $i - 1$ , “]=1)goto L_”,  $\text{high}(v)$ 
9:   if  $\text{flip}(v)$  then
10:    print “else {ret_b = !ret_b;”
11:    print “goto L_”,  $\text{low}(v)$ , “;}”
12:   else
13:    print “else goto L_”,  $\text{low}(v)$ 
14:    $W \leftarrow \text{Translate}(\rho, \text{high}(v), W)$ 
15:    $W \leftarrow \text{Translate}(\rho, \text{low}(v), W)$ 
16: return  $W$ 

```

Details of Function Translate (Algorithm 4): Given inputs ρ, v, W , Algorithm 4 performs a recursive graph traversal of $G^{(\rho_v)}$ as follows.

The C code block for internal node v is generated in lines 3 and 7–13. The block consists of a label `L_v`: and an `if-then-else` C construct. Note that label `L_v` univocally identifies the C code block related to node v . This may be implemented by printing the hexadecimal value of a pointer to v .

The `if-then-else` C construct is generated so as to traverse node v in graph $G^{(\rho_v)}$ in the following way. In line 8 the check `x[i-1]=1` is generated, being i such that $\text{var}(v) = x_i$. The code to take the then edge of v is also generated. Namely, it is sufficient to generate a `goto` statement to the C code block related to node $\text{high}(v)$. In lines 10–11 and 13 the code to take the else edge is generated, in the case `x[i-1]=1` is false. In this case, if the else edge is complemented, i.e., $\text{flip}(v)$ holds (lines 10–11), it is necessary to complement `ret_b` and then perform a `goto` statement to the C code block related to node $\text{low}(v)$ (lines 10–11). Otherwise, it is sufficient to generate a `goto` statement to the C code block related to node $\text{low}(v)$ (line 13).

Thus, the block generated for an internal node v , for proper i, l and h , has one of the following forms, depending on $\text{flip}(v)$:

- `L_v: if (x[i - 1]) goto L_h; else goto L_l;`
- `L_v: if (x[i-1]) goto L_h; else {ret_b = !ret_b; goto L_l;}`

There are two base cases for the recursion of function *Translate*:

- $v \in W$ (line 1), i.e., v has already been translated into a C code block as above. In this case, the set of visited COBDD nodes W is directly returned (line 1) without generating any C code. This allows us to retain COBDD node sharing;
- $v = 1$ (line 4), i.e., the terminal node 1 has been reached. In this case, the C code block to be generated is simply `L_1: return ret_b;`. Note that such a block will be generated only once.

In all other cases, function *Translate* ends with the recursive calls on the then and else edges (lines 14–15). Note that the visited nodes set W passed to the second recursive call is the result of the first recursive call. Correctness of function *Translate* is proved in Lemma 5.

D. An Example of Translation

Consider the COBDD ρ shown in Figure 2. Within ρ , consider `mgo` $K(x_0, x_1, x_2, u_0, u_1) = \llbracket 0x17, 1 \rrbracket$. By applying *SolveFunctionalEq*, we obtain $f_1(x_0, x_1, x_2) = \llbracket 0x15, 1 \rrbracket$ and $f_2(x_0, x_1, x_2) = \llbracket 0x10, 1 \rrbracket$. Note that `0xe` is shared between $G^{(\rho_{0x15})}$ and $G^{(\rho_{0x10})}$. Finally, by calling *GenerateCCode* (see Algorithm 3) on f_1, f_2 , we have the C code in Figure 3.

```

int K_bits(int *x, int action) {
  int ret_b;
  switch(action) {
    case 0: ret_b = 0; goto L_0x15;
    case 1: ret_b = 0; goto L_0x10;
  }
L_0x15:
  if (x[0] == 1) goto L_0x13;
  else { ret_b = !ret_b; goto L_0x14; }
L_0x13:
  if (x[1] == 1) goto L_0xe;
  else { ret_b = !ret_b; goto L_1; }
L_0xe:
  if (x[2] == 1) goto L_1;
  else { ret_b = !ret_b; goto L_1; }
L_0x14:
  if (x[1] == 1) goto L_0xe;
  else goto L_1;
L_0x10:
  if (x[0] == 1) goto L_0xe;
  else { ret_b = !ret_b; goto L_0xf; }
L_0xf:
  if (x[1] == 1) goto L_0xe;
  else { ret_b = !ret_b; goto L_0xe; }
L_1:
  return ret_b;
}

void K(int *x, int *u) {
  int i;
  for(i = 0; i < 2; i++)
    u[i] = K_bits(x, i);
}

```

Figure 3. C code for the mgo in Figure 2 as generated by Synthesize

VII. TRANSLATION PROOF OF CORRECTNESS

In this section, we prove the correctness of our approach (Theorem 6). That is, we show that the function K we generate indeed implements the given controller K , thus resulting in a correct-by-construction control software.

We begin by stating four useful lemmata for our proof. Lemma 2 is useful to prove Lemma 3, i.e., to prove correctness of function *SolveFunctionalEq*.

Lemma 2. Let $K : \mathbb{B}^n \times \mathbb{B}^r \rightarrow \mathbb{B}$ and let f_1, \dots, f_r be such that $f_i(\mathbf{x}) = \exists u_{i+1}, \dots, u_r K(\mathbf{x}, f_1(\mathbf{x}), \dots, f_{i-1}(\mathbf{x}), 1, u_{i+1}, \dots, u_r)$ for all $i \in [r]$. Then, $\mathbf{x} \in \text{Dom}(K) \Rightarrow K(\mathbf{x}, f_1(\mathbf{x}), \dots, f_r(\mathbf{x})) = 1$.

Proof: Let $\mathbf{x} \in \mathbb{B}^n$ be such that $\mathbf{x} \in \text{Dom}(K)$, i.e., $\exists \mathbf{u} K(\mathbf{x}, \mathbf{u}) = 1$. We prove the lemma by induction on r . For $r = 1$, we have $f_1(\mathbf{x}) = K(\mathbf{x}, 1)$. If $f_1(\mathbf{x}) = 1$, we have $K(\mathbf{x}, f_1(\mathbf{x})) = K(\mathbf{x}, 1) = f_1(\mathbf{x}) = 1$. If $f_1(\mathbf{x}) = 0$, we have $K(\mathbf{x}, f_1(\mathbf{x})) = K(\mathbf{x}, 0)$, and $K(\mathbf{x}, 0) = 1$ since $\mathbf{x} \in \text{Dom}(K)$ and $K(\mathbf{x}, 1) = 0$.

Suppose by induction that for all $\tilde{K} : \mathbb{B}^n \times \mathbb{B}^{r-1} \rightarrow \mathbb{B}$ $\tilde{K}(\mathbf{x}, \tilde{f}_1(\mathbf{x}), \dots, \tilde{f}_{r-1}(\mathbf{x})) = 1$, where for all $i \in [r-1]$ $\tilde{f}_i(\mathbf{x}) =$

$\exists u_{i+1}, \dots, u_{r-1} \tilde{K}(\mathbf{x}, \tilde{f}_1(\mathbf{x}), \dots, \tilde{f}_{i-1}(\mathbf{x}), 1, u_{i+1}, \dots, u_{r-1})$. We have that $\mathbf{x} \in \text{Dom}(\tilde{K})$ implies that either $\mathbf{x} \in \text{Dom}(\tilde{K}|_{u_1=0})$ or $\mathbf{x} \in \text{Dom}(\tilde{K}|_{u_1=1})$. Suppose $\mathbf{x} \in \text{Dom}(\tilde{K}|_{u_1=1})$ holds. We have that $\tilde{K}|_{u_1=1}(\mathbf{x}, \tilde{f}_2(\mathbf{x}), \dots, \tilde{f}_r(\mathbf{x})) = 1$, where for all $i = 2, \dots, r$ $\tilde{f}_i(\mathbf{x}) = \exists u_{i+1}, \dots, u_r \tilde{K}|_{u_1=1}(\mathbf{x}, \tilde{f}_2(\mathbf{x}), \dots, \tilde{f}_{i-1}(\mathbf{x}), 1, u_{i+1}, \dots, u_r)$. By construction, we have that $\tilde{f}_1(\mathbf{x}) = 1$ and $\tilde{f}_i(\mathbf{x}) = \tilde{f}_i(\mathbf{x})$ for $i \geq 2$, thus $1 = \tilde{K}|_{u_1=1}(\mathbf{x}, \tilde{f}_2(\mathbf{x}), \dots, \tilde{f}_r(\mathbf{x})) = \tilde{K}(\mathbf{x}, \tilde{f}_1(\mathbf{x}), \dots, \tilde{f}_r(\mathbf{x}))$. Analogously, if $\mathbf{x} \notin \text{Dom}(\tilde{K}|_{u_1=1}) \wedge \mathbf{x} \in \text{Dom}(\tilde{K}|_{u_1=0})$ we have that $\tilde{f}_1(\mathbf{x}) = 0$ and $\tilde{f}_i(\mathbf{x}) = \tilde{f}_i(\mathbf{x})$ for $i \geq 2$, thus $1 = \tilde{K}|_{u_1=0}(\mathbf{x}, \tilde{f}_2(\mathbf{x}), \dots, \tilde{f}_r(\mathbf{x})) = \tilde{K}(\mathbf{x}, \tilde{f}_1(\mathbf{x}), \dots, \tilde{f}_r(\mathbf{x}))$. ■

Lemma 3 states correctness of function *SolveFunctionalEq* of Algorithm 2.

Lemma 3. Let $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD with $\mathcal{V} = \mathcal{X} \cup \mathcal{U}$, $v \in V$ be a node, $b \in \mathbb{B}$ be a flipping bit. Let $\llbracket v, b \rrbracket = K(\mathbf{x}, \mathbf{u})$ and $r = |\mathcal{U}|$. Then function *SolveFunctionalEq*(ρ, v, b) (see Algorithm 2) outputs nodes v_1, \dots, v_r and boolean values b_1, \dots, b_r such that for all $i \in [r]$ $\llbracket v_i, b_i \rrbracket = f_i(\mathbf{x})$ and $\mathbf{x} \in \text{Dom}(K)$ implies $K(\mathbf{x}, f_1(\mathbf{x}), \dots, f_r(\mathbf{x})) = 1$.

Proof: Correctness of functions *COBDD_APP* and *COBDD_EX* (and lemma hypotheses) implies that for all $i \in [r]$ $f_i(\mathbf{x}) = \exists u_{i+1}, \dots, u_r K(\mathbf{x}, f_1(\mathbf{x}), \dots, f_{i-1}(\mathbf{x}), 1, u_{i+1}, \dots, u_r)$. By Lemma 2 we have the thesis. ■

Let *Translate_dup* be a function that works as function *Translate* of Algorithm 4, but that does not take nodes sharing into account. Function *Translate_dup* may be obtained from function *Translate* by deleting line 1 (highlighted in Algorithm 4) and by replacing calls to *Translate* in lines 14 and 15 with recursive calls to *Translate_dup* (with no changes on parameters). Lemma 4 states correctness of function *Translate_dup*.

Lemma 4. Let $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD, $v \in V$ be a node, $b \in \mathbb{B}$ be a flipping bit, and $W \subseteq V$ be a set of nodes. Then function *Translate_dup*(ρ, v, W) generates a sequence of labeled C statements $B_1 \dots B_k$ such that $k \geq |V_v|$ and for all $w \in V_v$: 1) label \perp_w is in B_i for some i and 2) starting an execution from label \perp_w with $\forall i \in [n] x[i-1] = x_i$ and $\text{ret_b} = \bar{b}$, if $\llbracket w, b \rrbracket = f_{w,b}$ then a return ret_b ; statement is invoked in at most $O(p)$ steps with $\text{ret_b} = f_{w,b}(\mathbf{x})$ and $p = \text{height}(w)$.

Proof: We prove this lemma by induction on v . Let $v = \mathbf{1}$, which implies $\llbracket v, b \rrbracket = \bar{b}$ and $V_v = \{\mathbf{1}\}$. We have that function *Translate_dup*(ρ, v, W) generates a single block B_1 (thus $k = 1 = |V_v|$) such that $B_1 = \perp_{\mathbf{1}}: \text{return ret_b}$; (lines 3–5 of Algorithm 4). Since by hypothesis we have $\text{ret_b} = \bar{b}$, and since starting from B_1 the return

statement is invoked in $O(1)$ steps, the base case of the induction is proved.

Let v be an internal node with $\text{var}(v) = x_i$ and let $f(\mathbf{x}) = \llbracket v, b \rrbracket$. Since $w \in V_v$ if and only if $w = v \vee w \in V_{\text{high}(v)} \vee w \in V_{\text{low}(v)}$, by induction hypothesis we only have to prove the thesis for $w = v$. We have that $f(\mathbf{x}) = x_i \llbracket \text{high}(v), b \rrbracket + \bar{x}_i \llbracket \text{low}(v), b \oplus \text{flip}(v) \rrbracket$, i.e., $f(\mathbf{x}) = x_i \llbracket \text{high}(v), b \rrbracket + \bar{x}_i \llbracket \text{low}(v), b \rrbracket$ if $\text{flip}(v) = 0$ and $f(\mathbf{x}) = x_i \llbracket \text{high}(v), b \rrbracket + \bar{x}_i \llbracket \text{low}(v), \bar{b} \rrbracket$ if $\text{flip}(v) = 1$. Since $f(\mathbf{x}) = x_i f|_{x_i=1}(\mathbf{x}) + \bar{x}_i f|_{x_i=0}(\mathbf{x})$, by Theorem 1 we have that $\llbracket \text{high}(v), b \rrbracket = f|_{x_i=1}(\mathbf{x})$, and that $\llbracket \text{low}(v), b \rrbracket = f|_{x_i=0}(\mathbf{x})$ if $\text{flip}(v) = 0$ and $\llbracket \text{low}(v), \bar{b} \rrbracket = f|_{x_i=0}(\mathbf{x})$ if $\text{flip}(v) = 1$.

By lines 3 and 8–13 of Algorithm 4, we have that function $\text{Translate_dup}(\rho, v, W)$ generates blocks $BB_{11} \dots B_{1h} B_{21} \dots B_{2l}$ such that $B = \text{L}_v$: if $(x[i-1] == 1)$ goto L_v ; else B_E where B_E is either goto L_v ; if $\text{flip}(v) = 0$ or $\{\text{ret_b} = !\text{ret_b}; \text{goto } \text{L}_v;\}$ if $\text{flip}(v) = 1$, and $B_{11} \dots B_{1h}$ ($B_{21} \dots B_{2l}$) are generated by the recursive call $\text{Translate_dup}(\rho, \text{high}(v), W)$ in line 14 ($\text{Translate_dup}(\rho, \text{low}(v), W)$ in line 15). By induction hypothesis and the above reasoning, if the execution starts at label L_v and $\text{ret_b} = \bar{b}$, then a return $\text{ret_b};$ statement is invoked in at most $O(p-1)$ steps with $\text{ret_b} = f|_{x_i=1}(\mathbf{x})$. As for the else case, we have that starting from L_v with $\text{ret_b} = \bar{b}$ ($\text{ret_b} = \bar{b}$) if $\text{flip}(v) = 0$ ($\text{flip}(v) = 1$), then a return $\text{ret_b};$ statement is invoked in at most $O(p-1)$ steps with $\text{ret_b} = f|_{x_i=0}(\mathbf{x})$. By construction of block B , starting from label L_v , a return $\text{ret_b};$ statement is invoked in at most $O(p-1+1) = O(p)$ steps with $\text{ret_b} = x_i f|_{x_i=1}(\mathbf{x}) + \bar{x}_i f|_{x_i=0}(\mathbf{x}) = f(\mathbf{x})$. Finally, note that by induction hypothesis $h \geq |V_{\text{high}(v)}|$ and $l \geq |V_{\text{low}(v)}|$, thus we have that $k = 1 + h + l \geq 1 + |V_{\text{high}(v)}| + |V_{\text{low}(v)}| \geq |V_v|$. ■

Lemma 5 extends Lemma 4 by also considering nodes sharing, thus stating correctness of function GenerateCCCode of Algorithm 3 and function Translate of Algorithm 4.

Lemma 5. Let $\rho = (V, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD and $v_1, \dots, v_r \in V$ be r nodes and $b_1, \dots, b_r \in \mathbb{B}$ be r flipping bits. Then lines 6–8 of function $\text{GenerateCCCode}(\rho, v_1, b_1, \dots, v_r, b_r)$ generate a sequence of labeled C statements $B_1 \dots B_k$ such that $k = |\cup_{i=1}^r V_{v_i}|$ and for all $v \in \cup_{i=1}^r V_{v_i}$: 1) the label L_v is in B_j for some j and 2) starting an execution from label L_v with $\forall j \in [n] x[j-1] = x_j$ and $\text{ret_b} = \bar{b}$, if $\llbracket v, b \rrbracket = f_{v,b}$ then a return $\text{ret_b};$ statement is invoked in at most $O(p)$ steps with $\text{ret_b} = f_{v,b}(\mathbf{x})$ and $p = \text{height}(w)$.

Proof: We begin by proving that $k = |\cup_{i=1}^r V_{v_i}|$. To this aim, we prove that for each node $v \in \cup_{i=1}^r V_{v_i}$, a unique block B_v is generated. This follows by how the nodes set W is managed by function Translate in lines 1–3

of Algorithm 4 and by function GenerateCCCode in lines 6–8 of Algorithm 3. In fact, function Translate , when called on parameters ρ, v, W , returns a set $W' \supseteq W$, and function GenerateCCCode calls Translate by always passing the W resulting by the previous call. Since a block is generated for node v only if v is not in W , and v is added to W only when a block is generated for node v , this proves this part of the lemma.

As for correctness, we prove this lemma by induction on m , being m the number of times that the return $W;$ statement in line 1 of Algorithm 4 is executed. As base of the induction, let $m = 1$ and let ρ, v, W be the parameters of the recursive call executing the first return $W;$ statement. Then, by construction of function Translate , v has been added to W in some previous recursive call with parameters ρ, v, \tilde{W} . In this previous recursive call, a block B_v with label L_v has been generated. Moreover, for this previous recursive call, thus for parameters ρ, v, \tilde{W} , we are in the hypothesis of Lemma 4, which implies that the induction base is proved.

Suppose now that the thesis holds for the first m executions of the return $W;$ statement in line 1 of Algorithm 4. Then, by construction of function Translate , v has been added to W in some previous recursive call with parameters ρ, v, \tilde{W} . In this previous recursive call, a block B_v with label L_v has been generated. Let $w_1, W_1, \dots, w_m, W_m$ be such that the m recursive calls executing the return $W;$ statement have parameters ρ, v_i, W_i (note that they are not necessarily distinct). By induction hypothesis, for all $i \in [m]$ starting from label L_v with $\forall j \in [n] x[j-1] = x_j$ and $\text{ret_b} = \bar{b}$, a return $\text{ret_b};$ statement is invoked in at most $O(p)$ steps with $\text{ret_b} = f_{w_i,b}(\mathbf{x})$. By Lemma 4 and its proof, the same holds for all $v \in V_v \setminus \{w_1, \dots, w_m\}$, thus it holds for all $v \in V_v$. ■

Finally, Theorem 6 states and proves correctness for function Synthesize of Algorithm 1.

Theorem 6. Let $\rho = (V, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD with $\mathcal{V} = \mathcal{X} \cup \mathcal{U}$, $v \in V$ be a node, $b \in \mathbb{B}$ be a boolean. Let $\llbracket v, b \rrbracket = K(\mathbf{x}, \mathbf{u})$, $r = |\mathcal{U}|$ and $n = |\mathcal{X}|$. Then function $\text{Synthesize}(\rho, v, b)$ generates a C function $\text{void } K(\text{int } *x, \text{int } *u)$ with the following property: for all $\mathbf{x} \in \text{Dom}(K)$, if before a call to $K \forall i \in [n] x[i-1] = x_i$, and after the call to $K \forall i \in [r] u[i-1] = u_i$, then $K(\mathbf{x}, \mathbf{u}) = 1$.

Furthermore, function K has WCET $\sum_{i=1}^r O(\text{height}(v_i))$, being v_1, \dots, v_r the nodes output by function SolveFunctionalEq .

Proof: Let $\mathbf{x} \in \text{Dom}(K)$ (i.e., $\exists \mathbf{u} K(\mathbf{x}, \mathbf{u}) = 1$) and suppose that for all $j \in [n] x[j-1] = x_j$. By lines 9–11 of Algorithm 3, for all $i \in [r]$, $u[i-1]$ will take the value returned by $K_bits(\mathbf{x}, i)$. In turn, by lines 3 and 4 of Algorithm 3, each $K_bits(\mathbf{x}, i)$ sets ret_b

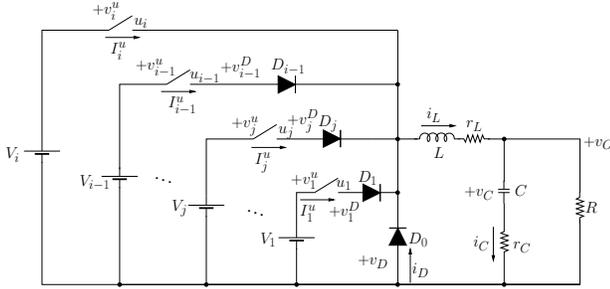


Figure 4. Multi-input Buck DC-DC converter.

to \bar{b}_i and makes a jump to label L_{v_i} . By Lemma 3 and by construction of *Synthesize*, such b_1, \dots, b_r and v_1, \dots, v_r are such that that $\llbracket v_1, b_1 \rrbracket = f_1(\mathbf{x}), \dots, \llbracket v_r, b_r \rrbracket = f_r(\mathbf{x})$ and $K(\mathbf{x}, f_1(\mathbf{x}), \dots, f_r(\mathbf{x})) = 1$. By Lemma 5, the sequence of calls $K_bits(\mathbf{x}, 1), \dots, K_bits(\mathbf{x}, r)$ will indeed return, in at most $\sum_{j=1}^r O(\text{height}(v_i))$ steps, $f_1(\mathbf{x}), \dots, f_r(\mathbf{x})$.

Corollary 7. Let $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD with $\mathcal{V} = \mathcal{X} \cup \mathcal{U}$, $v \in V$ be a node, $b \in \mathbb{B}$ be a boolean. Let $\llbracket v, b \rrbracket = K(\mathbf{x}, \mathbf{u})$, $r = |\mathcal{U}|$ and $n = |\mathcal{X}|$. Then the C function K output by function *Synthesize*(ρ, v, b) has WCET $O(rn)$.

Proof: The corollary immediately follows from Theorem 6 and from the fact that, for all $v \in V$, $\text{height}(v) \leq n$.

VIII. EXPERIMENTAL RESULTS

We implemented our synthesis algorithm in C programming language, using the CUDD (Colorado University Decision Diagram [19]) package for OBDD based computations and BLIF (Berkeley Logic Interchange Format [20]) files to represent input OBDDs. We name the resulting tool KSS (*Kontrol Software Synthesizer*). KSS is part of a more general tool named QKS (*Quantized feedback Kontrol Synthesizer* [6]).

A. Experimental Settings

We present experimental results obtained by using KSS on given COBDDs ρ_1, \dots, ρ_5 such that for all $i \in [5]$ ρ_i represents the mgo $K_i(\mathbf{x}, \mathbf{u})$ for a buck DC/DC converter with i inputs.

The *multi-input* buck DC-DC converter [21] in Figure 4 is a mixed-mode analog circuit converting the DC input voltage (V_i in Figure 4) to a desired DC output voltage (v_O in Figure 4). As an example, buck DC-DC converters are used off-chip to scale down the typical laptop battery voltage (12-24) to the just few volts needed by the laptop processor (e.g., see [22]) as well as on-chip to support *Dynamic Voltage and Frequency Scaling* (DVFS) in multicore processors (e.g., see [23]). Because of its widespread use, control schemas

Table I
KSS PERFORMANCES

r	CPU	MEM	$ K $	$ F^{unsh} $	$ Sw $	%
1	3.0e-02	1.0e+08	12137	2646	2646	0.0e+00
2	1.1e-01	1.3e+08	25848	5827	5076	1.3e+01
3	1.7e-01	1.8e+08	36430	10346	8606	1.7e+01
4	2.5e-01	2.4e+08	46551	15004	12285	1.8e+01
5	3.6e-01	3.3e+08	65835	21031	16768	2.0e+01

for buck DC-DC converters have been widely studied. The typical software based approach (e.g., see [22]) is to control the switches u_1, \dots, u_i in Figure 4 (typically implemented with a MOSFET, i.e., a metal-oxide-semiconductor field-effect transistor [24]) with a microcontroller.

In the following experiments, we fix $n = |\mathbf{x}| = 20$ and we have that $r_i = |\mathbf{u}| = i$. Finally, K_i is an intermediate output of the QKS tool described in [6].

For each ρ_i , we run KSS so as to compute *Synthesize*(ρ_i, v_i, b_i) (see Algorithm 1), being $\llbracket v_i, b_i \rrbracket = K_i(\mathbf{x}, \mathbf{u})$. In the following, we will call $\langle v_{1i}, b_{1i}, \dots, v_{ii}, b_{ii} \rangle$, with $v_{ji} \in V_i, b_{ji} \in \mathbb{B}$, the output of function *SolveFunctionalEq*(ρ_i, v_i, b_i). Moreover, we call $f_{1i}, \dots, f_{ii} : \mathbb{B}^n \rightarrow \mathbb{B}$ the i boolean functions such that $\llbracket v_{ji}, b_{ji} \rrbracket = f_{ji}(\mathbf{x})$. All our experiments have been carried out on a 3.0 GHz Intel hyperthreaded Quad Core Linux PC with 8 GB of RAM.

B. KSS Performance

In this section, we will show the performance (in terms of computation time, memory, and output size) of the algorithms discussed in Section VI. Table I show our experimental results. The i -th row in Table I corresponds to experiments running KSS so as to compute *Synthesize*(ρ_i, v_i, b_i). Columns in Table I have the following meaning. Column r shows the number of action variables $|\mathbf{u}|$ (note that $|\mathbf{x}| = 20$ on all our experiments). Column *CPU* shows the computation time of KSS (in secs). Column *MEM* shows the memory usage for KSS (in bytes). Column $|K|$ shows the number of nodes of the COBDD representation for $K_i(\mathbf{x}, \mathbf{u})$, i.e., $|V_{v_i}|$. Column $|F^{unsh}|$ shows the number of nodes of the COBDD representations of f_{1i}, \dots, f_{ii} , without considering nodes sharing among such COBDDs. Note that we do consider nodes sharing inside each f_{ji} separately. That is, $|F^{unsh}| = \sum_{j=1}^i |V_{v_{ji}}|$ is the size of a trivial implementation of f_{1i}, \dots, f_{ii} in which each f_{ji} is implemented by a stand-alone C function. Column $|Sw|$ shows the size of the control software generated by KSS, i.e., the number of nodes of the COBDD representations f_{1i}, \dots, f_{ii} , considering also nodes sharing among such COBDDs. That is, $|Sw| = |\cup_{j=1}^i V_{v_{ji}}|$ is the number of C code blocks generated by lines 6–8 of function *GenerateCCCode* in Algorithm 3. Finally, Column % shows the gain percentage we obtain by considering nodes sharing among COBDD representations

for f_{1i}, \dots, f_{ii} , i.e., $(1 - \frac{|Sw|}{|F_{unsh}|})100$.

From Table I we can see that, in less than 1 second and within 350 MB of RAM we are able to synthesize the control software for the multi-input buck with $r = 5$ action variables, starting from a COBDD representation of K with about 6.6×10^4 nodes. The control software we synthesize in such a case has about 1.7×10^4 lines of code, whilst a control software not taking into account COBDD nodes sharing would have had about 2.1×10^4 lines of code. Thus, we obtain a 20% gain towards a trivial implementation.

IX. CONCLUSION AND FUTURE WORK

In this paper, we presented an algorithm which, starting from a boolean relation K representing the set of implementations meeting the given system specifications, generates a correct-by-construction C code implementing K . This entails finding boolean functions F such that $K(x, F(x)) = 1$ holds, and then implement such F . WCET for the generated control software is linear linear in nr , being r the number of functions in F and $n = |x|$. Furthermore, we formally proved that our algorithm is correct.

We implemented our algorithm in a tool named KSS. Given our algorithm properties explained above, by using KSS it is possible to synthesize correct-by-construction control software, provided that K is provably correct with respect to initial formal specifications. This is the case in [6], thus this methodology, e.g., allows to synthesize correct-by-construction control software starting from formal specifications for DTLHSs. We have shown feasibility of our proposed approach by presenting experimental results on using it to synthesize C controllers for a multi-input buck DC-DC converter.

The WCET of the resulting control software may be too high for some systems in which nr is high, or for which the control software has to provide actions with an high frequency. In order to speed-up the WCET, a natural possible future research direction is to investigate how to parallelize the generated control software.

ACKNOWLEDGMENTS

This work has received funding both from MIUR project TRAMP and the FP7/2007-2013 project ULISSE (grant agreement n°218815).

REFERENCES

- [1] F. Mari, I. Melatti, I. Salvo, and E. Tronci, "From boolean relations to control software," in *ICSEA 2011*, pp. 528–533.
- [2] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. on Computers*, vol. C-35, no. 8, 1986, pp. 677–691.
- [3] D. Baneres, J. Cortadella, and M. Kishinevsky, "A recursive paradigm to solve boolean relations," *IEEE Trans. on Computers*, vol. 58, no. 4, 2009, pp. 512–527.
- [4] R. Wille and R. Drechsler, "Bdd-based synthesis of reversible logic for large functions," in *DAC 2009*, pp. 270–275.
- [5] E. Tronci, "Automatic synthesis of controllers from formal specifications," in *ICFEM 1998*, pp. 134–143.
- [6] F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Synthesis of quantized feedback control software for discrete time linear hybrid systems," in *CAV 2010*, ser. LNCS 6174, pp. 180–195.
- [7] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. The MIT Press, 1999.
- [8] F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Quantized feedback control software synthesis from system level formal specifications for buck dc/dc converters," *CoRR*, vol. abs/1105.5640, 2011.
- [9] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, A. Sangiovanni-Vincentelli, E. Sentovich, and K. Suzuki, "Synthesis of software programs for embedded control applications," *IEEE Trans. CAD*, vol. 18, 1995, pp. 834–849.
- [10] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Formal synthesis of embedded control software: Application to vehicle management systems," in *AIAA Infotech@Aerospace, 2011*.
- [11] A. Cimatti, M. Roveri, and P. Traverso, "Strong planning in non-deterministic domains via model checking," in *AIPS 1998*, pp. 36–43.
- [12] V. Alimuzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci, "On model based synthesis of embedded control software," in *EMSOFT 2012*, pp. 227–236.
- [13] A. Pnueli and R. Rosner, "On the synthesis of an asynchronous reactive module," in *ICALP 1989*, pp. 652–671.
- [14] A. Girault and É. Rutten, "Automating the addition of fault tolerance with discrete controller synthesis," *Formal Methods in System Design*, vol. 35, no. 2, 2009, pp. 190–225.
- [15] —, "From boolean functional equations to control software," *CoRR*, vol. abs/1106.0468, 2011.
- [16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.
- [17] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a bdd package," in *DAC 1990*, pp. 40–45.
- [18] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient boolean function manipulation," in *DAC 1990*, pp. 52–57.
- [19] "CUDD Web Page," <http://vlsi.colorado.edu/fabio/CUDD>, last accessed 20th dec 2012
- [20] "Berkeley logic interchange format (BLIF)," http://bear.ces.cwru.edu/eecs_cad/sis_blif.pdf, last accessed 20th dec 2012.

- [21] M. Rodriguez, P. Fernandez-Miaja, A. Rodriguez, and J. Sebastian, "A multiple-input digitally controlled buck converter for envelope tracking applications in radiofrequency power amplifiers," *IEEE Trans. on Power Electronics*, vol. 25, no. 2, 2010, pp. 369–381.
- [22] W.-C. So, C. Tse, and Y.-S. Lee, "Development of a fuzzy logic controller for dc/dc converters: design, computer simulation, and experimental evaluation," *IEEE Trans. on Power Electronics*, vol. 11, no. 1, 1996, pp. 24–32.
- [23] W. Kim, M. S. Gupta, G.-Y. Wei, and D. M. Brooks, "Enabling on-chip switching regulators for multi-core processors using current staggering," in *ASGI 2007*.
- [24] Y. Cheng and C. Hu, *MOSFET Modeling and Bsim3 User's Guide*. Kluwer Academic Publishers, 1999.

Dynamic Reverse Engineering of Graphical User Interfaces

Inês Coimbra Morgado and Ana C. R. Paiva
 Department of Informatics Engineering,
 Faculty of Engineering, University of Porto,
 rua Dr. Roberto Frias, 4200-465 Porto, Portugal
 {pro11016, apaiva}@fe.up.pt

João Pascoal Faria
 Department of Informatics Engineering,
 Faculty of Engineering, University of Porto
 rua Dr. Roberto Frias, 4200-465 Porto, Portugal
 INESC TEC, Porto, Portugal
 jpf@fe.up.pt

Abstract—This paper presents a dynamic reverse engineering approach and a tool, ReGUI, developed to reduce the effort of obtaining models of the structure and behaviour of a software applications Graphical User Interface (GUI). It describes, in more detail, the architecture of the REGUI tool, the process followed to extract information and the different types of models produced to represent such information. Each model describes different characteristics of the GUI. Besides graphical representations, which allow checking visually properties of the GUI, the tool also generates a textual model in Spec# to be used in the context of model based GUI testing and a Symbolic Model Verification model, which enables the verification of several properties expressed in computation tree logic. The models produced must be completed and validated in order to ensure that they faithfully describe the intended behaviour. This validation process may be performed by manually analysing the graphical models produced or automatically by proving properties, such as reachability, through model checking. A feasibility study is described to illustrate the overall approach, the tool and the results obtained.

Keywords—ReGUI; Dynamic Reverse Engineering; GUI Testing; Properties Verification; CTL; Model Checking; SMV

I. INTRODUCTION

This paper extends the research work presented in [1], which describes a reverse engineering tool to extract a model from the execution of a Graphical User Interface (GUI). In particular, the state of the art is improved and the overall approach is described in more detail. Moreover, a new module was implemented in order to automatically generate a Symbolic Model Verification (SMV) model for model checking. The case study was extended in order to illustrate the additional features.

GUI models are key inputs for several advanced techniques, such as Model Based GUI Testing (MBGT) [2], [3], [4], which enables the automatic test case generation, increasing the systematisation and automation of the GUI testing process, and model checking, which enables an automatic verification and validation of some properties of the system. However, the manual construction of such models is a time consuming and error prone activity. One way of diminishing this effort is to automatically construct part of the model by a reverse engineering process.

The challenge tackled in this research work is the automatic construction of part of the software's GUI model (structure and behaviour) using a dynamic reverse engineering technique. The extracted information is presented in several formats that allow performing different types of analysis, such as, visual inspection, model checking and MBGT.

The term *reverse engineering* was firstly defined in 1985 by Rekoff [5] as “*the process of developing a set of specifications for a complex hardware system by an orderly examination of specimens of that system*”. Five years later, Chikofsky and Cross [6] adapted this definition to software systems: “*Reverse Engineering is the process of analysing a subject system to (1) identify the system's components and interrelationships and (2) to create representations of the system in another form or at a higher level of abstraction*”.

The origin of software reverse engineering lies on the necessity of improving and automating software maintenance. It is estimated that program comprehension [7], *i.e.*, understanding the structure and the behaviour of the software, corresponds to over 50% of software maintenance [8]. As such, developing tools which may aid software engineers on this task is of the utmost importance. Reverse engineering has already proved to be useful on this subject. For example, reverse engineering helped coping with the Y2K problem, with the European currency conversion and with the migration of information systems to the web and towards the electronic commerce [9]. For such reasons, the IEEE-1219 standard¹, which was replaced by the IEEE-14764 one², recommends reverse engineering as a key supporting technology to software maintenance [9].

In the last two decades, reverse engineering tools have evolved considerably and, nowadays, reverse engineering is useful for other fields of study rather than software maintenance, such as, software testing and auditing security and vulnerability.

According to Canfora *et al.* [10], nowadays, the main goals of reverse engineering are:

- recovering architectures and design patterns;

¹IEEE Standard for Software Maintenance

²Standard for Software Engineering - Software Life Cycle Processes - Maintenance

- re-documenting programs and databases;
- identifying reusable assets;
- building traceability between software artefacts;
- computing change impacts;
- re-modularising existing systems;
- renewing user interfaces;
- migrating towards new architectures and platforms;
- testing and maintenance.

As every technology, reverse engineering techniques can also be used with malicious intent [11], like removing software protection and limitations or allowing unauthorised access to systems/data. However, the developers may use the same techniques in order to assure software's safety.

Yet in 1990, Chikofsky and Cross [6] divided the reverse engineering process in two parts: an *analyser*, which collects and stores the information, and an *abstractor*, which represents that information at a higher level of abstraction, *i.e.*, a model, either graphical or textual. Figure 1 depicts the representation of a common reverse engineering process.

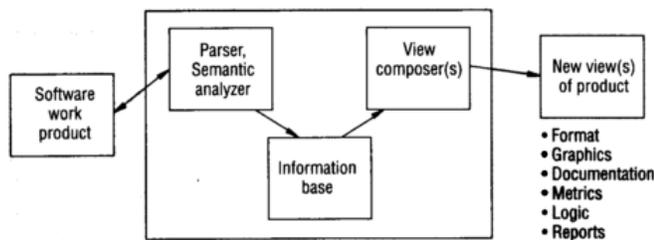


Fig. 1. Model of reverse engineering tools' architecture [6]

In Figure 1 the analyser is referred to as a *Parser, Semantic analyzer* because, in the early years of reverse engineering, these were the most common techniques. Nowadays, besides static techniques [12], which extract information from the source code, there are other three different approaches [13], [14] for reverse engineering: dynamic, which extracts information from a running software system; hybrid, which mixes both static and dynamic approaches; and historical, which extracts information about the evolution of the system from version control systems, like SVN³ or GIT⁴. Dynamic approaches have the advantages of being independent of the GUI implementation language and of not requiring access to source code. However, the existing dynamic approaches have limitations in terms of the behavioural information they are able to extract.

This paper describes a dynamic reverse engineering tool, ReGUI, developed to automatically extract structural and behavioural information from a GUI, including some behavioural information not obtained by other tools, like dependencies between GUI controls. ReGUI also distinguishes from other tools by producing multiple views and formats of the gathered information for different kinds of analysis: visual models

enable a visual inspection of some properties, such as the number of windows of the GUI; textual models can be used for MBGT (Spec# [15] model) and to prove properties (SMV [16] model).

ReGUI v2.0 is fully automatic and uses a different approach from its previous version [17] so the results achieved, such as the extracted dependencies and the produced graphs, are different.

The rest of this paper is organised as follows. Section II presents the state of the art on user interface reverse engineering. Section III presents the proposed approach and the developed tool, ReGUI. Section IV presents the exploration process and the challenges faced during the development of ReGUI. Section V describes the outputs that can be obtained. Section VI presents a feasibility study on *Microsoft Notepad v6.1*, presenting the results obtained. Section VII presents some conclusions about this research work, along with the limitations of the approach and future work.

II. STATE OF THE ART

This Section presents the state of the art on reverse engineering, mainly on GUI reverse engineering, regarding the time interval from 2000 to 2011.

A. Static Analysis

As stated in Section I, static reverse engineering extracts information from textual sources, usually the source code, of the Application Under Analysis (AUA) [12]. The main techniques used in static analysis are code parsers, which are used to analyse the source code itself, query engines, which are used to verify certain facts of the code, and presentation engines, which are used to depict the query results [18].

Staiger's Approach: Staiger [19] presented, in 2007, an approach for the Bauhaus tool suite⁵ [20] to statically reverse engineer the source code of a GUI, in order to support program understanding, maintenance and standards' analysis. Staiger claimed researchers had not focused their work on the static analysis of GUIs, even though most applications provided one. Staiger's approach is divided in three different phases: detecting the GUI elements, detecting widget hierarchies and detecting event connections. For the first phase, Staiger detects which data types on the source code had any connection to the GUI. Having detected these data types, he identifies the variables and respective parameters, as well as the functions or methods that are part of the GUI. After the source code elements are identified, the next phase finds the actual GUI elements, by identifying when each of the elements was created and the relationships among them. This provides the hierarchy of the elements. The third phase of the approach detects the different event handlers, the event they are handling and the element which triggers them. Along the execution, a window graph is generated containing all the extracted information on the several windows of the GUI. This approach

³svn.apache.org

⁴git-scm.com

⁵<http://www.bauhaus-stuttgart.de/>

was intended for C/C++ applications with a GUI implemented with GUI libraries, such as GTK⁶ or Qt⁷.

Lutteroth's Approach: Lutteroth [21] presented, in 2008, an approach whose goal was to automatically improve the layout of hard-coded GUIs. The approach extracts the GUI's structure and transforms it into a formal layout, Auckland Layout Model (ALM), which was defined by Lutteroth and Weber [22] in 2006. This approach extracts the structure of the GUI, by identifying its root element and navigating through its descendants. During this process, the position of each of the elements is mapped to a *tabstop*, which is a ALM property that represents a position in the coordinate system of a GUI.

Afterwards, the properties of each element, such as size and position, are updated, according to what best fits the GUI. For example, if an element has static content, a button for instance, then its size remains unaltered; otherwise, the best size is calculated according to its possible contents. In the end, the obtained layout may even be automatically improved. This may be done, for example, with the aid of some layout standards.

GUISurfer: In 2010, Silva *et al.* [23] developed the GUISurfer framework to test GUI-based Java applications, by following a static reverse engineering approach. The framework is composed by three tools: *File Parser*, *ASTAnalyser* and *Graph*. The first tool is responsible for the reverse engineering process, by parsing the GUI's source code and extracting behavioural information into an Abstract Syntax Tree (AST) [24]. Then, the second tool slices the information contained in the AST, focusing on the interface layer. In order to do so, the *ASTAnalyser* requires, besides the AST, the entry point of the application (the main method) and the set of GUI elements, which are part of the slicing process. In the end of this second phase, two files are generated: one containing the initial state of the GUI and one containing the events that may occur from the initial state. The third tool processes these two files and generates two Haskell specification files, which map the different events and conditions to actions on the GUI.

B. Dynamic Analysis

This Section describes existing dynamic reverse engineering approaches without and with code instrumentation.

1) Approaches Without Instrumentation:

GUIRipper: In 2003, Memon *et al.* [25] presented GUIRipper, a dynamic reverse engineering tool, which extracts behavioural information from the GUI of Java systems for testing purposes [26].

GUIRipper automatically interacts with the system's GUI, attempting to open as many windows as it can and, during this process, it extracts the GUI's structure and behaviour, producing three different artefacts. The GUI Forest is a graph representing the structure of the GUI. Each node represents a window of the GUI, containing the structure of its elements; an edge from a node *a* to a node *b* indicates that the window

represented by the node *b* is accessible from the window represented by the node *a*. The second artefact is an event flow graph (EFG), which represents the behaviour of the GUI. Each node represents an event, such as *click on the button OK*; an edge from a node *a* to a node *b* indicates event *b* can follow event *a*. The third artefact is an integration tree, which relates the different components of the GUI. This last artefact is necessary to rip off the GUI into several components, generating EFGs for each one.

Amalfitano et al.'s Approach: Amalfitano *et al.*'s [27] presented, in 2008, an approach to reverse engineer Ajax [28] based Rich Internet Applications (RIAs) [29] as they claimed the problematic of modelling and validating this type of applications had not yet been explored thoroughly. They intended to fill this gap by dynamically extracting the behaviour of an application and representing it as a finite state machine (FSM) [30].

The analyser runs the RIA under analysis within a controlled environment and an event analysis takes place, *i.e.*, information on the sequence of events is extracted. The state chart diagram depicted in Figure 2 models this first phase, which includes two main states: waiting for an event to occur (*Event Waiting*) and waiting for an event handler to be complete (*Event Handling Completion Waiting*). Whenever an event is raised, information such as the type of the event, at what time it occurred and on which element it occurred is recorded. This process begins when the application starts.

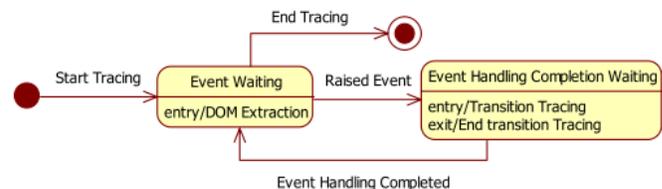


Fig. 2. The trace activity for the extraction step [27]

The second phase, abstraction, is composed by three steps. Initially, the information is transformed into a graph (a transition graph), which models the flow of client interfaces. The second step consists on using a clustering technique to analyse the transition graph in order to group equivalent nodes and edges. The clustering is based on the evaluation of several interface equivalence criteria, such as, the DOM structures including the same set of active element nodes and offering the same interaction behaviour to the users. This way, the issue of state explosion is dealt with. Finally, the FSM is generated with each state corresponding to each node of the clustered transition graph. This approach was validated by the development of a Java tool, RE-RIA, which implements both phases of the process.

2) Approaches With Instrumentation: There are some dynamic techniques that require source code (or byte code) instrumentation. Code instrumentation consists in inserting logging code into the existing one. As this is achieved during

⁶www.gtk.org

⁷qt.digia.com/

run time and without access to the code, they are considered dynamic approaches instead of hybrid ones.

Briand et al.'s Approach: Briand *et al.* proposed, in 2006 [31], an approach to dynamically extract behavioural information from Java distributed communications, namely Remote Method Invocation applications. The extracted information is represented as UML sequence diagrams. Even though there may be several applications to these diagrams, they intended to test the consistency of the code with the design.

Briand *et al.* divided their approach in two phases. The first phase consists in the instrumentation of the source code. In order to make their approach as little intrusive as possible, Briand *et al.* used Aspect Oriented Programming [32]. The second phase analyses the execution traces, creates the corresponding models and transforms them into scenario diagrams.

As in every dynamic analysis strategy, the extracted information is limited to the extent of the system's exploration and to the context in which each action was executed. This way, Briand *et al.* defined two meta-models: one to describe the information extracted from the execution traces and another to describe what they called scenario diagrams, which are UML sequence diagrams but limited to the context (scenario) of the execution. In order to transform the first meta-model into the second, they defined rules in the Object Constraint Language.

Finally, Briand *et al.* claimed that one of the biggest advantages of their approach was the usage of meta-models and transformation rules as these are formalised and can be easily improved and compared to others.

Safyallah and Sartipi: In 2006, Safyallah and Sartipi [33] presented an approach to identify the features of a system by identifying sequential patterns in the execution traces of the system. In order to do so, Safyallah and Sartipi divided their approach into two phases. In the first phase, the execution traces are extracted. This is achieved by setting scenarios, which are based in the domain of the application, the documentation and the familiarity of the user with the system, to examine each feature, and by source code instrumentation (inserting the name of the function at the beginning and at the end of each of them). Executing the scenarios provided the execution traces. In the second phase, a sequential pattern mining algorithm was applied to the extracted traces in order to obtain the most frequent sequential patterns. Figure 3 depicts the type of patterns identified: with this type of analysis, it is possible to identify, for example, that a *lock* is eventually followed by an *unlock*.

```

1  ABCDEAXBC
2  AGXBC
3  AXBC

```

Fig. 3. Sequential pattern: the sequence *ABC* is repeated [34]

This enabled the identification of generic functionalities (common to the different features of the system) and the ones that were feature-specific. With this approach, Safyallah and

Sartipi were able to ease program comprehension and feature to source code assignment.

Alafi's Approach: In 2009, Alafi [35] also presented an approach and a tool (PHP2XMI), which intended to extract behavioural information by instrumentation of the source code of the AUA and analysing the generated event traces. The ultimate goal of this approach is to ease the security analysis and testing of PHP-based⁸ web applications.

The PHP2XMI tool functions in three steps. The first step corresponds to the code instrumentation. This step enables the extraction of information on page *URLs*, *http* variables, sessions and cookies. The second step executes the application, generating the execution traces, which are then filtered to ignore redundant information, and storing the relevant information in a SQL database. The third and final step transforms the stored data into UML 2.1 sequence diagrams [36]. These diagrams can be depicted by any UML 2.1 tool set.

C. Hybrid Analysis

Hybrid analysis provides an improvement of the completeness, scope and precision of the extraction as it mixes both static and dynamic approaches, trying to maximise the amount of extracted information [13]. This Section presents some of the works that follow this line of research.

Systä's Approach: In 2000, in her dissertation, Systä [37] presented an approach combining the advantages of both static and dynamic analyses, with special focus on the dynamic part, for reverse engineering a Java software system.

The static part consists in parsing the system's byte code with a byte code extractor in order to extract the system's structure. This information is represented as a graph, which can be visualised with the Rigi reverse engineering environment, developed by Müller *et al.* [38]. Afterwards, the system is run under a customised jdk debugger, JDebugger, producing dynamic event trace information, control flow data, which was represented as scenario diagrams. These diagrams could be visualised with the SCED dynamic modelling tool [39], which transforms them into a single state diagram.

Systä developed a prototype, Shimba, which applies the described approach, integrating the Rigi system with the SCED tool. Without disregarding the other applications of the extracted information, debugging is presented as being a very useful one.

Frank et al.'s Approach: In 2001, Frank *et al.* [40] presented an approach to dynamically reverse engineer a mobile application for Android, iOS or Java ME. They extract a model of the life cycle, which can be used to detect errors, like verifying if an application's information is saved when it has to be interrupted, *e.g.*, save the text of an e-mail when an incoming call occurs. Even though the reverse engineering process itself is processed during run-time, it is necessary to previously alter the source code, which makes this a hybrid approach.

Frank *et al.*'s approach was divided in four phases. The first and second ones are of the responsibility of the developer as

⁸<http://www.php.net/>

they consist in programming the life cycle's code, overwriting every call-back method called in life cycle changes, and inserting logging code to all the overwritten methods. In the third phase, black-box tests [41] are applied to the mobile application in order to identify the different triggers. For this, it is of the utmost importance that the first two phases have been processed thoroughly. In the fourth phase, the information extracted in the previous phase is used to derive an application's life cycle model and to identify properties of the application at certain states of the life cycle. The model is a state diagram of the application. The states can be, for example, *running*, *paused*, *background*. The transitions are labelled according to the corresponding actions, e.g., *onCreate()* and *onStop()*. This model may be useful in several contexts, such as verifying the consistency of the application's life cycle or identifying properties of the application at a given state.

D. Discussion

Apart from the work of Lutteroth *et al.* [21], which only extracts structural information from GUIs to improve layout, the analysed reverse engineering approaches are similar to ours because they extract both structural and behavioural information. However the purpose of each approach may be different: program comprehension (for testing and/or maintenance) [19], [23], [25], [27], [31], [42]; debugging [37]; properties verification [40]; feature to source code mapping [33]; and security analysis [35]. The purpose of the presented approach is program comprehension and properties verification.

Regarding the information extracted, there are similarities regarding structural information (GUI elements, their properties and hierarchy relations) the set of approaches extract but varieties regarding behavioural information. Some approaches extract events (their handlers and the relations between them) [19], [23], [25], [27]; sequence of actions [27], [31], [33], [35], [37]; sequential patterns [33]; and lifecycle of the system [40]. The presented approach extracts structural information, alike the remaining approaches, and behavioural information on navigation and on dependencies between the different GUI controls.

Another characteristic that distinguishes the several studied approaches is the representation (abstractor) of the extracted information. Most of the approaches represent their information in only one structure: sequence diagrams [31], [35], [37]; state diagrams [27], [40]; specification file [21]; graphs [19] or sequence patterns [33]. There are only two approaches which opt to represent the information in more than one way: Silva *et al.* [23] extracted both a specification file and an AST and Memon *et al.* [25] extracted a window graph and an event flow graph. As far as the authors know, there is only one approach that enables verification of properties [40], but it focuses strictly on the life cycle of mobile applications.

In addition, most of the approaches can only be applied to one platform (web [27], [35] or mobile [40]), or to one language (Java [23], [25], [31], [37] or C/C++ [19]). The approach described in this paper uses UI Automation that allows

extracting information from desktop and web applications, which increases the range of supported platforms.

III. REGUI OVERVIEW

The goal of this research work is to diminish the effort of producing visual and formal models of the GUI of a software application. The approach followed is the extraction of the information from the GUI under analysis by a dynamic reverse engineering approach. This way, this approach is independent of the programming language in which the GUI was written, broadening its applicability. This Section presents an overview of the proposed approach.

A. Architecture and Outputs

Figure 4 depicts an overview of the approach proposed in this paper.

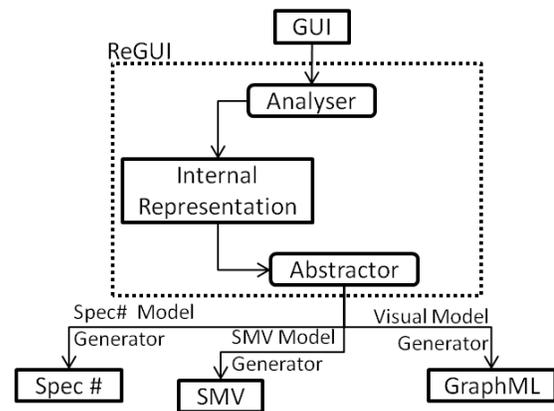


Fig. 4. Architecture and outputs obtained with the ReGUI tool

The analyser is responsible for the exploration of the GUI and extraction of the structural and behavioural information. The exploration process is discussed in more detail in Section IV-A. The abstractor is responsible for representing the extracted information in different ways: visual models, which enable a quick visual inspection; a Spec# model, which can be used for MBGT; and a SMV model, which enables the automatic verification of properties. These models are explained in detail in Section V.

B. Extracted Information

Figure 5 represents the information extracted by ReGUI. *GUI Elements* can be *Windows* or *Controls* that may be initially enabled or disabled. *Windows* may be *modal* (in which case it is not possible to interact with other windows of the same application while this one is opened) or *modeless* (it is possible to interact with other windows). *Windows* are composed of *Controls*, which may be *menu items* or others. The elements in the diagram (classes and relationships) are annotated with graphical symbols used in the visual models generated by ReGUI.

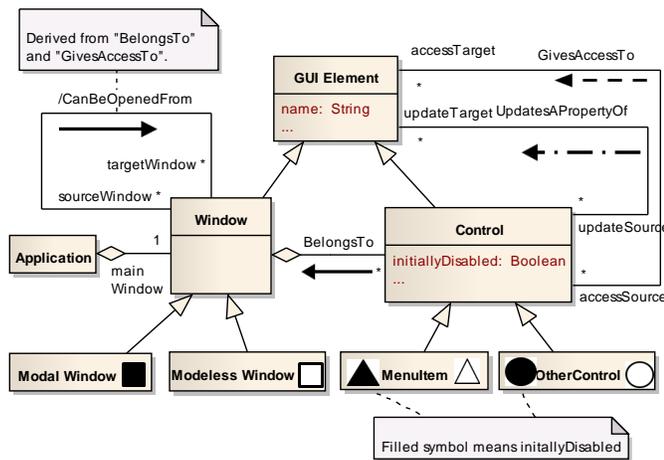


Fig. 5. Annotated metamodel of the models generated by the ReGUI tool (see Figure 4)

The associations between the objects represent the extracted behaviour. When interacting with a control, there are five possible identifiable outcomes:

- *open* - a window is opened;
- *close* - a window is closed;
- *expansion* - new controls become accessible. For instance, the expansion of a menu;
- *update* - one or more properties of one or more elements are updated. For instance, the name of a window is modified or an enabled control becomes disabled (or *vice-versa*);
- *skip* - nothing happens.

The first three outcomes are represented by the *GivesAccessTo* relation, while the third one is represented by the *UpdatesAPropertyOf* relation. If a control of a window (*BelongsTo* relation) opens another window (*GivesAccessTo* relation), then there is a *CanBeOpenedFrom* relation between the second and the first windows. This makes the last relation a derivation from the two previous ones.

C. Front-End

The ReGUI front-end is shown in Figure 6.



Fig. 6. ReGUI front-end

In order to start the extraction process, it is necessary to identify the GUI to be analysed. In order to do so, it is necessary to drag the *Spy Tool* symbol and drop it on top of the GUI. Following, the user must press the button *Play*, which will start the exploration process. The name of this button changes to *Playing* during the execution. At the end of the execution, all models except the Spec# and the SMV ones have already been generated. As such, the user may press the *Generate SMV Model* button in order to generate the SMV model, as well as, the *Generate Spec# Model* button in order to generate the Spec# model.

IV. REGUI ANALYSER

In this Section, the analyser process is described, namely the exploration process and the challenges tackled during the implementation.

A. Exploration Process

In general, a dynamic exploration can be classified according to its automation, manual or automatic, and to whether or not it is guided. If the exploration is automatic not being guided means it is random whilst a guided exploration would require some heuristic to determine which control should be explored at each instance. If, otherwise, it is manual, being guided means the user actions are driven by a particular goal, whilst not being guided would just indicate the user has complete freedom in choosing the next control.

Given the goal of this approach, the exploration must be performed automatically, remaining the choice of being guided or not. If the random exploration took long enough, eventually the entire interface would be explored. However, the amount of time a company has is limited and, thus, this approach is not ideal. On the other hand, the guided exploration depends completely on the algorithm used for the exploration.

As in most situations, the best solution would be to get the advantages of each approach: follow a guided approach mixed with a random one and, if and when the exploration hits a breakpoint, the tool should ask the user to interact with the GUI in order to move forward with the automatic exploration. ReGUI follows a guided exploration based on the order of the elements.

The exploration process is divided in two phases. First, ReGUI navigates through every menu option in order to extract the initial state of the GUI, *i.e.*, which GUI elements are enabled/disabled at the beginning of the execution, in the main window. For the second phase, ReGUI navigates through all the menus and interacts with the ones enabled at that instance. After each interaction, ReGUI verifies if any window has opened. If so, ReGUI extracts its structure, closing it afterwards. Following, ReGUI goes through all the menus again in order to verify if any state changed, *i.e.*, if a previously enabled element became disabled or *vice-versa*. All the information extracted is organised in internal structures, which are described in Section V.

In order to interact with the GUI, ReGUI uses UI Automation [43], which is the accessibility framework for Microsoft

Windows, available on all operating systems that support Windows Presentation Foundation. This framework represents all the applications opened in a computer as a tree (a *Tree Walker*), whose root is the *Desktop* and whose nodes are the applications opened at a certain moment. The GUI elements are represented as nodes, children of the application to which they belong. In the UI Automation framework each of these elements is an *Automation Element*.

B. Challenges

During the development of ReGUI, it was necessary to face some challenges:

1) *Identification of GUI elements*: GUI elements may have dynamic properties, *i.e.*, properties which may vary along the execution, such as the *RunTimeIdProcess* and the *Name*, and do not have a property which uniquely identifies them. During the exploration process, the identification of an element is performed by comparing its properties with the ones of other elements. There are some that, when used for comparing two elements, undoubtedly distinguish them when their values are different. For example, if two controls are a *button* and a *menu item*, then they are necessarily different. However, this sort of properties may not be sufficient. As such, an heuristic based on some properties was implemented to compare two elements: an element *a* is considered to be the same as an element *b* when it is the one which most resembles element *b*, considering a minimum threshold. The properties to be used in the comparison can be configured in the beginning of the execution.

2) *Exploration order*: In general, the extracted information depends on the order in which the GUI is explored. Currently, ReGUI follows a depth-first algorithm, *i.e.*, all the options of a menu are explored before exploring the next one. The exploration of the children of a node follows the order in which they appear on the GUI. However, if the exploration followed a different order, the dependencies extracted would be different. An example of such may be found in the *Microsoft Notepad v6.1* application and is depicted in Figure 7. The menu item *Select All* requires the presence of text in the main window in order to produce any results. Since there is no text in the main window, in the beginning, interacting with this menu item does not have any effect. However, after interacting with the *Time/Date* menu item, which writes the time and date in the main window, the *Select All* menu item would produce visible results: it would select the text, enabling the menu items *Cut*, *Copy* and *Delete* and disabling the *Select All* menu item itself.

3) *Synchronisation*: To automatically interact with a GUI, it is necessary to wait for the interface to respond after each action. In order to surpass this problem, ReGUI checks (with event handlers) when any changes occurred in the UI Automation tree (which reflects the state of the screen in each moment) and continues after that. For example, after expanding a menu, its submenus are added to the *UI Automation* tree as its children, launching an event. The event handler catches it and ReGUI acts accordingly. When verifying whether or not a window opened, there is an event handler similar to the one

used to catch a menu expansion. However, when invoking an element for the first time, there is no way of previously knowing if any event will occur. This way, after invoking an element, ReGUI waits either for the event handler to catch the event or for a defined amount of time.

4) *Closing a Window*: During the execution it is necessary to close windows that are eventually opened, in order to continue the exploration process. However, there is no standard way of closing them. Windows usually have a top right button for closing purposes but, when this is not available, ReGUI looks for one of these buttons to close it: *cancel*, *no*, *close*, *ok*, *continue* or *x*.

V. REGUI ABTRACTOR

ReGUI generates different views on the extracted information. Each of these views represents different aspects of the structure and behaviour of the GUI under analysis, enabling a rapid visual inspection of such aspects. The current views ReGUI is able to extract are a tree representing the structure of the GUI and the hierarchy between the different elements, and four graphs representing its behaviour. Every node of these four graphs corresponds to a node in the tree. The information stored in these structures is used to generate the formal models both in Spec# and in SMV. The next sub-sections describe these different outputs, explaining the type of information represented in each of them. The Figures referred along this Section are examples of outputs and can be depicted along Section VI.

A. Structural Information: ReGUI Tree

The ReGUI tree merges all the UI Automation trees produced during the exploration process. Initially, the ReGUI tree has only the elements visible at the beginning of the exploration and, at the end, it has every element which has become visible at some point of the exploration, such as the content of the windows opened along the process and sub-menu options. An examples is depicted in Figure 12.

B. Behavioural Information

Extracting behavioural information is useful for different purposes, such as modelling the GUI behaviour, generating test cases, proving properties or usability analysis. This Section describes the different views generated by ReGUI on the behavioural information extracted.

1) *Navigation Graph*: The navigation graph represents the nodes relevant to the navigation, *i.e.*, this graph stores information about which user actions must be performed in order to open the different windows of the application. A visual representation of this graph is depicted in Figure 13. A solid edge between a window *w1* and a GUI element *e1* means *e1* is inside of *w1* whilst a dashed edge between two GUI elements *e1* and *e2* means *e2* becomes accessible after interacting with *e1*.

Figure 8 is a subset of Figure 5 of Section III and depicts the information extracted by ReGUI that is represented in this graph, as well as the graphical symbols used.

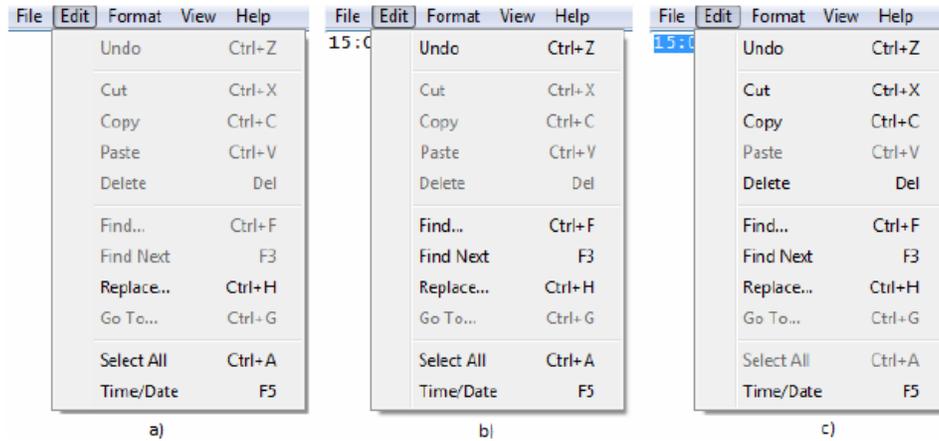


Fig. 7. Menu item *Edit* on *Microsoft Notepad v6.1*: a) after invoking the menu item *Select All* and before invoking the menu item *Time/Date*; b) after invoking the menu item *Time/Date*; c) after invoking again the menu item *Select All*

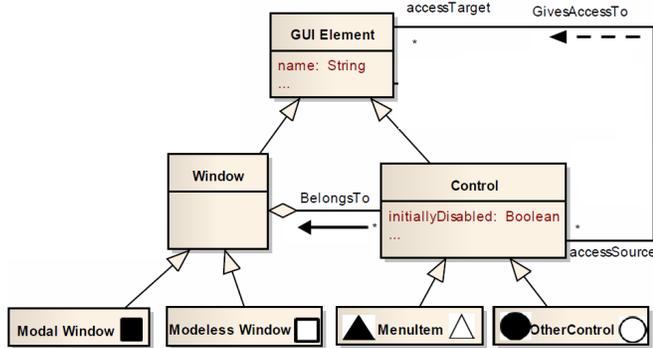


Fig. 8. Representation of the different elements and their relationships in the navigation graph

2) *Window Graph*: The window graph shows a subset of the information represented by the navigation graph. It describes the windows that may be opened in the application. Figure 14 is a visual representation of this graph. A window may be modal or modeless. An edge between two nodes $w1$ and $w2$ means that it is possible to open window $w2$ by interacting with elements of window $w1$.

Figure 9 is a subset of Figure 5 of Section III and depicts the information extracted by ReGUI that is represented in this graph.

3) *Disabled Graph*: The disabled graph's purpose is to show which nodes are accessible but disabled in the beginning of the execution (obtained during the first phase of the exploration process described in Section IV-A). The enabled property of an element may vary during the second phase but that modification is not represented in this graph. An example of this graph is depicted in Figure 15. The nodes correspond to some GUI elements, being filled when disabled and empty when enabled. A solid edge between two nodes $n1$ and $n2$ means that $n2$ belongs to $n1$. On the other hand, a dashed edge

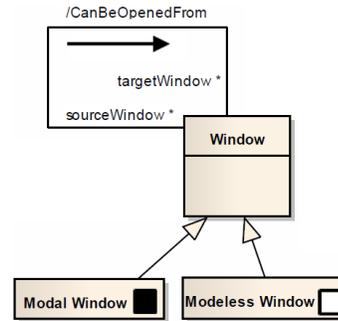


Fig. 9. Representation of the different elements and their relationships in the window graph

between those nodes means $n2$ is accessible after interacting with $n1$.

Figure 8 is also applicable to this graph as the relations between the controls have the same meaning, even though the information represented in both graphs is different.

4) *Dependency Graph*: A dependency between two elements A and B means that interacting with A modifies the value of a property of B . An example of a dependency would be if interacting with A enabled a previously disabled B . Figure 16 is the visual representation of a dependency graph obtained during an exploration process. A solid edge between a window $w1$ and a node $n1$ means $n1$ is accessed from $w1$ and a dashed edge between two nodes $n1$ and $n2$ means there is a dependency between $n1$ and $n2$.

Figure 10 is a subset of Figure 5 of Section III and depicts the information extracted by ReGUI that is represented in this graph.

C. Spec# Model

Spec# is a formal specification language that can be used as input to the model-based testing tool Spec Explorer [44], for automatic test generation.

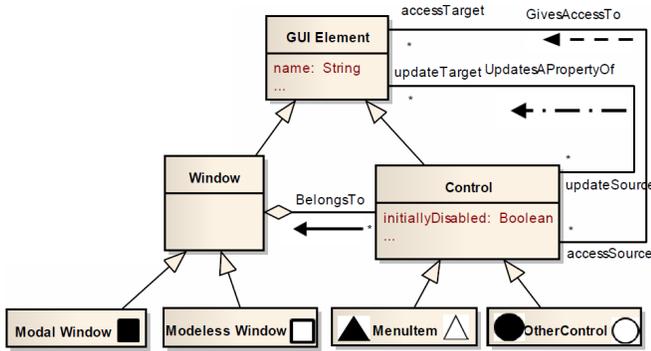


Fig. 10. Representation of the different elements and their relationships in the dependency graph

The Spec# model is obtained by applying the rules in Figure 11 on the navigation graph. Each window generates a namespace and each edge generates a method annotated with *[Action]*. Action methods in Spec# are methods that will be used as steps within the following generated test cases. Methods without annotations are only used internally. All the elements relevant to the navigation are represented as variables (*var*) having three possible values: 1, if the element is accessible and enabled; 2, if the element is accessible but disabled; and 3, if the element is not accessible. At the beginning of the execution, the only possibly accessible elements are the ones belonging to the main window, as every other window is not accessible itself. Every variable and method corresponding to elements belonging to a certain window must be placed under the namespace representing that window. An example of a model generated by the application of these rules is in Figure 17.

D. SMV Model

After obtaining a model one problem that may rise is how to verify properties on it. This may be tackled by model checking techniques, which verify if given properties are valid on the model under analysis. The verification of properties can be very useful, for example, in usability analysis and improvement [45], [46]. The one used in this approach is symbolic model checking [47]. Properties are expressed in Computation Tree Logic, which is a propositional temporal logic, and the system is modelled as a FSM.

Some of the properties that can usually be verified are *reachability*, i.e., if it is possible to reach every node, *liveness*, i.e., under certain circumstances, something will eventually happen, *safety*, i.e., under some circumstances, something will never happen, *fairness*, i.e., under certain conditions, something will always happen, and *deadlock-freeness*, i.e., the system does not get into a cycle from which it cannot come out.

With this approach, the state machine representing the system is generated automatically based on the navigation graph. Alike this graph, each state represents a GUI element, which is represented by a unique *id*. The first state corresponds

```

Rule 1 □ - windowName
Spec#:
namespace windowName;
var windowName = 1; //if main window
var windowName = 3; //if other windows

Rule 2 □ → Δ / ▲
Spec#:
windowName MenuOptionName
// Apply Rule 1 to WindowName
var menuOptionName = 1; //if Δ and
//main window
var menuOptionName = 2; //if ▲ and
//main window
var menuOptionName = 3; //if other windows
[Action] MenuOptionName()
requires menuOptionName == 1; { }

Rule 3 □ → ○
WindowName ButtonName
Spec#:
// Apply Rule 1 to WindowName
var buttonName = 1; //if main window
var buttonName = 3; //if other windows
[Action] ButtonName()
requires buttonName == 1; { }

Rule 4 Δ -----▶ Δ / ▲
MenuOptionName1 MenuOptionName2
Spec#:
var menuOptionName2 = 3;
[Action] void MenuOptionName1()
requires menuOptionName1 == 1; {
  menuOptionName2 = 1; //or 2 if ▲
}
[Action] void MenuOptionName2()
requires menuOptionName2 == 1; { }

Rule 5 Δ -----▶ □
MenuOptionName WindowName
Spec#:
[Action] void MenuOptionName()
requires menuOptionName == 1; {
  WindowName.windowName = 1;
  menuOptionName = 3;
}
//Rule 1 for WindowName
//if not yet constructed

Rule 6 ○ -----▶ □
ButtonName WindowName
Spec#:
[Action] void ButtonName()
requires buttonName == 1; {
  WindowName.windowName = 1;
  buttonName = 3;
}
//Rule 1 for WindowName
//if not yet constructed
    
```

Fig. 11. Rules for the Spec# generation

to the main window, having *id* 1. The relations *belongsTo* of the navigation graph were eliminated for this representation because they do not describe user actions.

This model is imported to the SMV tool⁹ and is composed of three modules:

- *getInfo(id)*, where further information about the states may be represented. In this case, the type of GUI elements can be 1 for window, 2 for menu item and 3 for other controls;
- *getNextState(id)*, which represents information about the next state (e.g., how many and which states follow a given state);
- *main*, in which the state machine is described along with the specification of the properties to be verified.

Figure 18 depicts an example of the SMV description of a state machine.

VI. FEASIBILITY STUDY

In order to check and test the feasibility of the approach presented in Section IV, ReGUI was run on *Microsoft Notepad v6.1*. In this Section the different outputs resultant from this experiment are presented and analysed. The window, navigation, disabled and dependencies graphs were visualised with a template for Microsoft Excel, NodeXL¹⁰.

⁹<http://www.cs.cmu.edu/~modelcheck/smv.html>

¹⁰<http://nodexl.codeplex.com/>

enough for the tester to detect some abnormalities in the GUI's behaviour.

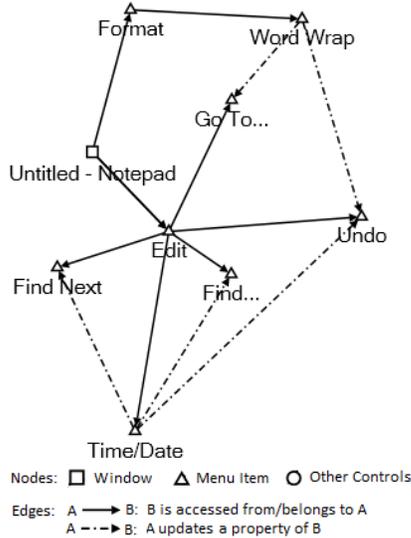


Fig. 16. Visual representation of the dependency graph

C. Spec# Model

Using the extracted information it is possible to obtain another kind of model: a Spec# model, which is a formal representation of the behaviour of the GUI. At this moment, the Spec # model only represents the information gathered on the navigation graph.

Figure 17 depicts a small sample of the generated Spec# model for this case study. The rules applied to generate this Spec# model, presented in Figure 11, are enumerated in comments (*//*). The first namespace corresponds to the main window of the *Notepad* software application. The two methods within this namespace describe the behaviour when interacting with the menu item *File* and with the menu item *Save*. The second namespace corresponds to the window *Save As* and its method describes the interaction with the button *Close* inside that window.

After validating and completing the model, it can be used, for example, as input for the MBGT approach described in [48].

D. SMV Model

In order to verify properties on the extracted information, it goes through a transformation process to SMV. Until now, only the information represented by the navigation graph is used to verify properties. The navigation graph is automatically transformed into a SMV state machine (see Figure 18).

Figure 18 depicts the representation of the state machine in the SMV model for this case study. The state machine has an initial state (*init(state)*) and transitions (*next(state)*). The meaning of each transition is described in comments (*--*). Three variables have been declared: *state*, which corresponds

```
namespace WindowUntitled__Notepad;
var windowUntitled__Notepad = 1;
var menu_itemFile = 1;
var menu_itemSave = 3;
[Action] void Menu_itemFile ()
  requires menu_itemFile == 1;{
  menu_itemSave = 1;
};
[Action] void Menu_itemSave ()
  requires menu_itemSave == 1;{
  menu_itemSave = 3;
  WindowSave_As.windowSave_As = 1;
};

namespace WindowSave_As;
var windowSave_As4 = 3;
var buttonClose = 3;
[Action] ButtonClose ()
  requires buttonClose == 1;{
  buttonClose = 3;
  WindowUntitled__Notepad.
  windowUntitled__Notepad = 1;
};
```

Fig. 17. Sample of the Spec# formal model generated

to the *id* of the control represented by that state; *follow*, which has two attributes: *state*, which corresponds to the set of possible next states, and *num*, which indicates the number of possible next states; and *moreInfo*, which has the attribute *type* that represents the type of the control corresponding to that state.

With this state machine, it is possible to verify several properties to evaluate, for instance, usability properties, such as:

- regardless of the current state, it is always possible to reach the main window (state = 1):
 $AF \text{ state} = 1;$
- check the presence of deadlocks:
 $!(EF (AG (follow.num = 1 \ \& \ state \ in \ follow.state)))$.
It checks if when there is only one out transition, the next state is different from the current state;
- regardless of the current state, it is possible to go back to the main window in *x* steps (three, e.g.):
 $EBF \ 1..3 \ \text{state} = 1;$
- when on the main window, there is always a window *x* steps away (three, e.g.):
 $state = 1 \ - \> \ ABF \ 1..3 \ moreInfo.tipo = 1.$

Running the SMV model for the *Microsoft Notepad* application, it is possible to state that:

- it is always possible to reach the main window, regardless of the state;
- no deadlocks were detected;
- it is always possible to get to the main window in three steps;
- there is always a window three steps away.

VII. CONCLUSIONS AND FUTURE WORK

ReGUI is capable of extracting important information about the behaviour of the AUA, such as navigational information

```

MODULE main
VAR
  state: 1..20;
  follow: getNextState(state);
  moreInfo: getInfo(state);

ASSIGN
  init(state) := 1;
  next(state) :=
    case
      state = 1: {2, 4, 6, 7, 9, 11, 13, 15, 17, 19};
      --from state 1, it is possible to go to states 2 (Open...),
      --4 (Save), 6 (Save As...), 7 (Page Setup), 9 (Print...),
      --11 (Replace...), 13 (Go To...), 15 (Font...),
      --17 (View Help), 19 (About Notepad)
      state = 2: 3; --goes to window Open
      state = 3: 1; --goes to the main window
      state = 4: 5; --goes to window Save As
      state = 5: 1; --goes to the main window
      state = 6: 5; --goes to window Save As
      state = 7: 8; --goes to window Page Setup
      state = 8: 1; --goes to the main window
      state = 9: 10; --goes to window Print
      state = 10: 1; --goes to the main window
      state = 11: 12; --goes to window Replace
      state = 12: 1; --goes to the main window
      state = 13: 14; --goes to window Go To Line
      state = 14: 1; --goes to the main window
      state = 15: 16; --goes to window Font
      state = 16: 1; --goes to the main window
      state = 17: 18; --goes to window Windows Help and Support
      state = 18: 1; --goes to the main window
      state = 19: 20; --goes to window About Notepad
      state = 20: 1; --goes to the main window
    esac;

```

Fig. 18. State machine in SMV

and which GUI elements become enabled or disabled after interacting with another element. The exploration process is fully automatic, with the user just having to point out the AUA.

The outputs generated by the ReGUI tool are extremely useful for program comprehension and for program verification as the graphs can be used to verify some important properties, such as reachability and deadlock-freeness, and the Spec# model can be used for test case generation and platform migration, for example. Even though the ReGUI tool does not generate the totality of the Spec# model, it already provides an important part of it.

The static and hybrid approaches have, by definition, a different purpose than the one presented in this paper as they require the source code, contrary to dynamic approaches. Comparing with other dynamic approaches, it is possible to conclude this approach extracts more information. Memon's approach [25] extracts information on the structure and the relation between the different events, which is represented by the *ReGUI Tree* and the navigation and window graphs, whilst this approach also extracts information on the dependency between the different controls. Similarly, Amalfitano's approach [27] is focused on the events (when they are raised and completed) and not on the dependency part. Briand *et al.* [31], Safyallah and Sartipi [33] and Alafi's [35] approaches require instrumentation, even though they are considered dynamic

approaches, which makes these approaches more intrusives than the approach presented in this paper. Moreover, the behavioural information extracted enables proving different properties through model checking. None of the analysed approaches provides such analysis.

The main difficulties faced during the development were the lack of GUI standards and the necessity of synchronisation. ReGUI has still some limitations. For instance, currently, it only supports interaction through the *invoke pattern* but it may evolve to interact through other patterns. In addition, it just tries to open windows from the main window and there are still other dependencies that may be explored.

The future work will be focused on solving these limitations, on improving the exploration of the GUI, *i.e.*, interact with the different controls more than once and in different orders and on improving the Spec# generation. It is also intended to apply this approach to other platforms, such as web and mobile.

VIII. ACKNOWLEDGEMENTS

This work is financed by the ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within the project FCOMP-01-0124-FEDER-020554 and the PhD scholarship SFRH/BD/81075/2011.

REFERENCES

- [1] I. Coimbra Morgado, A. Paiva, and J. Pascoal Faria. Reverse Engineering of Graphical User Interfaces. In *The Sixth International Conference on Software Engineering Advances (ICSEA '11)*, number c, pages 293–298, Barcelona, 2011.
- [2] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, November 2007.
- [3] Ana C. R. Paiva, João C. P. Faria, and Raul F. A. M. Vidal. Specification-based testing of user interfaces. In Joaquim A. Jorge, Nuno Jardim Nunes, and João Falcão e Cunha, editors, *10th International Workshop on Interactive Systems. Design, Specification, and Verification (DSV-IS '03)*, pages 139–153, Funchal, Portugal, 2003.
- [4] Ana C. R. Paiva, João C. P. Faria, and Pedro M. C. Mendes. Reverse engineered formal models for GUI testing. In *The 12th international conference on Formal methods for industrial critical systems*, pages 218–233, Berlin, Germany, July 2007. Springer-Verlag.
- [5] MG Rekoff. On Reverse Engineering. *IEEE Trans. Systems, Man, and Cybernetics*, (March-April):244 – 252, 1985.
- [6] E.J. Chikofsky and J.H. Cross. Reverse Engineering and Design Recovery: a Taxonomy. *IEEE Software*, 7(1):13–17, 1990.
- [7] Ted J. Biggerstaff, Bharat G. Mitbender, and Dallas Webster. The concept assignment problem in program understanding. In *The 15th international conference on Software Engineering (ICSE '93)*, pages 482–498, May 1993.
- [8] Thomas A. Standish. An Essay on Software Reuse. *IEEE Transactions on Software Engineering*, SE-10(5):494–497, September 1984.
- [9] Hausi A. Muller, Jens H. Jahnke, Dennis B. Smith, and Margaret-Anne Storey. Reverse engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering - ICSE '00*, pages 47–60, New York, New York, USA, May 2000. ACM Press.
- [10] Gerardo Canfora, Massimiliano Di Penta, and Luigi Cerulo. Achievements and challenges in software reverse engineering. *Communications of the ACM*, 54(4):142, April 2011.
- [11] Eldad Eilam. *Reversing: Secrets of Reverse Engineering*. Wiley, 2005.
- [12] David Binkley. Source Code Analysis: A Road Map. In *Future of Software Engineering (FOSE '07)*, pages 104–119. IEEE, May 2007.

- [13] Thoms Bell. The concept of dynamic analysis. *ACM SIGSOFT Software Engineering Notes*, 24(6):216–234, November 1999.
- [14] Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(2):77–131, March 2007.
- [15] Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec\# Programming System: An Overview. In *International Conference in Construction and Analysis of Safe, Secure and Interoperable Smart Devices (CASSIS '04)*, pages 49–69, Marseille, France, 2004. Springer.
- [16] Kenneth L. McMillan. *Getting Started with SMV*. Cadence Berkeley Labs, 2001 Addison St., Berkeley, CA, USA, 1999.
- [17] A.M.P. Grilo, A.C.R. Paiva, and J.P. Faria. Reverse engineering of GUI models for testing. In *The 5th Iberian Conference on Information Systems and Technologies (CISTI '10)*, number July, pages 1–6. IEEE, 2010.
- [18] Alexandru Telea, Lucian Voinea, and Heorhiy Byelas. Architecting an Open System for Querying Large C and C++ Code Bases. *S. African Computer Journal*, 41(December):43–56, 2008.
- [19] Stefan Staiger. Static Analysis of Programs with Graphical User Interface. In *11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, pages 252–264. IEEE, 2007.
- [20] Aoun Raza, Gunther Vogel, and Erhard Plöederer. Bauhaus A Tool Suite for Program Analysis and Reverse Engineering. In *Reliable Software Technologies, Ada Europe 2006*, page 71, 2006.
- [21] Christof Lutteroth. Automated reverse engineering of hard-coded GUI layouts. In *The 9th conference on Australasian user interface (AUIC '08)*, pages 65–73. ACM, January 2008.
- [22] Christof Lutteroth and Gerald Weber. User interface layout with ordinal and linear constraints. In *The 7th Australasian User Interface Conference (AUIC '06)*, pages 53–60, January 2006.
- [23] João Carlos Silva, Rui Gonçalo, João Saraiva, and José Creissac Campos. The GUISurfer Tool: Towards a Language Independent Approach to Reverse Engineering GUI Code. In *2nd ACM SIGCHI symposium on Engineering interactive computing systems*, pages 181–186, Berlin, 2010. ACM.
- [24] Nicola Howarth. Abstract Syntax Tree Design. Technical Report August 1995, Architecture Projects Management Limited, 1995.
- [25] Atif M. Memon, Ishan Banerjee, and Adithya Nagarajan. GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing. In *The 10th Working Conference on Reverse Engineering (WCRE '03)*, 2003.
- [26] Daniel R. Hackner and Atif M. Memon. Test case generator for GUITAR. In *Companion of the 13th international conference on Software engineering (ICSE Companion '08)*, ICSE Companion '08, page 959, New York, New York, USA, 2008. ACM Press.
- [27] Domenico Amalfitano, Anna Rita Fasolino, and Porfirio Tramontana. Reverse Engineering Finite State Machines from Rich Internet Applications. In *The 15th Working Conference on Reverse Engineering (WCRE '08)*, pages 69–73. IEEE, October 2008.
- [28] Jesse James Garrett. Ajax: A New Approach to Web Applications. *Adaptive Path*, 2005.
- [29] Cameron O'Rourke. A Look at Rich Internet Applications. *Oracle Magazine*, 2004.
- [30] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines—a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
- [31] L.C. Briand, Y. Labiche, and J. Leduc. Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software. *IEEE Transactions on Software Engineering*, 32(9):642–663, September 2006.
- [32] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, *The 11th European Conference on Object-Oriented Programming (ECOOP'97)*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin / Heidelberg, 1997.
- [33] H. Safyallah and K. Sartipi. Dynamic Analysis of Software Systems using Execution Pattern Mining. In *The 14th IEEE International Conference on Program Comprehension (ICPC '06)*, pages 84–88. IEEE, 2006.
- [34] Tao Xie, Suresh Thummalapenta, and D Lo. Data mining for software engineering. *IEEE Computer*, 42(8):55–62, 2009.
- [35] Manar H. Alalfi, James R. Cordy, and Thomas R. Dean. A verification framework for access control in dynamic web applications. In *The Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering (C3S2E '09)*, page 109, New York, New York, USA, May 2009. ACM Press.
- [36] Object Management Group. UML 2.1.2, 2012.
- [37] Tarja Systä. *Static and Dynamic Reverse Engineering Techniques for Java Software Systems*. Phd, University of Tampere, 2000.
- [38] Hausi A. Müller, Scott R. Tilley, and Kenny Wong. Understanding software systems using reverse engineering technology perspectives from the Rigi project. In *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering (CASCON '93)*, CASCON '93, pages Volume1: 217–226. IBM Press, 1993.
- [39] K. Koskimies, T. Systs, J. Tuomi, and T. Mannisto. Automated support for modeling OO software. *IEEE Software*, 15(1):87–94, 1998.
- [40] Dominik Franke, Corinna Elsemann, Stefan Kowalewski, and Carsten Weise. Reverse Engineering of Mobile Application Lifecycles. In *18th Working Conference on Reverse Engineering (WCRE '11)*, pages 283–292. IEEE, October 2011.
- [41] Glenford J. Myers. Art of Software Testing. March 1979.
- [42] Manar H. Alalfi, James R. Cordy, and Thomas R. Dean. Automated Reverse Engineering of UML Sequence Diagrams for Dynamic Web Applications. In *International Conference on Software Testing, Verification, and Validation Workshops (ICSTW '09)*, pages 287–294. IEEE, April 2009.
- [43] Rob Haverty. New accessibility model for Microsoft Windows and cross platform development. *SIGACCESS Access. Comput.*, (82):11–17, 2005.
- [44] Margus Veanes, Colin Campbell, Wolfgang Grieskamp, Wolfram Schulte, Nikolai Tillmann, Lev Nachmanson, Robert Hierons, Jonathan Bowen, and Mark Harman. Model-based testing of object-oriented reactive systems with spec explorer. In Robert M. Hierons, Jonathan P. Bowen, and Mark Harman, editors, *Formal Models and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 39–76. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
- [45] Fabio Paternò and Carmen Santoro. Integrating Model Checking and HCI Tools to Help Designers Verify User Interface Properties. In *7th International Workshop on Interactive Systems Design, Specification and Verification*, Limerick, Ireland, 2001.
- [46] Nadjat Kamel, Sid Ahmed Selouani, and Habib Hamam. A Model-Checking Approach for the Verification of CARE Usability Properties for Multimodal User Interfaces. *International Review on Computers & Software*, 4(1):152—160, 2009.
- [47] E Clarke, K McMillan, S Campos, and V Hartonas-Garmhausen. Symbolic model checking. In Rajeev Alur and Thomas Henzinger, editors, *Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 419–422. Springer Berlin / Heidelberg, 1996.
- [48] Ana C. R. Paiva, João C. P. Faria, Nikolai Tillmann, and Raul A. M. Vidal. A Model-to-implementation Mapping Tool for Automated Model-based GUI Testing. In *7th International Conference on Formal Engineering Methods (ICFEM '05)*, pages 450–464, 2005.

Event-Sequence Testing using Answer-Set Programming

Martin Brain¹, Esra Erdem², Katsumi Inoue³, Johannes Oetsch⁴, Jörg Pührer⁴, Hans Tompits⁴, and Cemal Yilmaz²

¹ University of Oxford, Department of Computer Science,
Oxford, OX1 3QD, UK

Email: martin.brain@cs.ox.ac.uk

²Sabanci University, Faculty of Engineering and Natural Sciences,
Orhanli, Tuzla, Istanbul 34956, Turkey

Email: {esraerdem,cyilmaz}@sabanciuniv.edu

³National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

Email: inoue@nii.ac.jp

⁴Technische Universität Wien, Institut für Informationssysteme 184/3,
Favoritenstraße 9-11, A-1040 Vienna, Austria

Email: {oetsch,puehrer,tompits}@kr.tuwien.ac.at

Abstract—In many applications, faults are triggered by events that occur in a particular order. In fact, many bugs are caused by the interaction of only a low number of such events. Based on this assumption, *sequence covering arrays* (SCAs) have recently been proposed as suitable designs for event sequence testing. In practice, directly applying SCAs for testing is often impaired by additional constraints, and SCAs have to be adapted to fit application-specific needs. Modifying precomputed SCAs to account for problem variations can be problematic, if not impossible, and developing dedicated algorithms is costly. In this article, we propose *answer-set programming* (ASP), a well-known knowledge-representation formalism from the area of artificial intelligence based on logic programming, as a declarative paradigm for computing SCAs. Our approach allows to concisely state complex coverage criteria in an *elaboration tolerant* way, i.e., small variations of a problem specification require only small modifications of the ASP representation. Employing ASP for computing SCAs is further justified by new complexity results related to event-sequence testing that are established in this work.

Keywords-event-sequence testing; complexity analysis; combinatorial interaction testing; answer-set programming.

I. INTRODUCTION

This article is an extension of a previous conference version [1]. Besides an extended discussion of related work, the first important extensions is a complexity analysis of the main computational problem which gives further justification of our solution approach. The second important extension is that we use a different problem encoding which is, on the one hand, simpler than the one used in the previous conference paper, and, on the the other hand, significantly improves many results.

In many applications, faults only show up if events occur in a certain order. An example are atomicity violations in multi-threaded applications where a pair of shared memory accesses

of one thread is interleaved with an unfortunate access of another thread. Testing such applications thus requires exercising *event sequences*. Since the number of event sequences is factorial in the number of events, exhaustive testing is infeasible in general. If we assume that bugs are triggered by the interaction of only a low number of events, testing costs can be reduced drastically without sacrificing much fault-detection potential by using suitable combinatorial designs [2], [3]. To this end, Kuhn et al. [4] introduced *sequence covering arrays* (SCAs) for combinatorial event sequence testing. An SCA of strength t is an array of permutations of events such that every ordering of any t events appears as a subsequence of at least one row. For illustration, the following matrix is an SCA for four events $\{1,2,3,4\}$ with $t = 3$:

$$\begin{pmatrix} 3 & 1 & 2 & 4 \\ 1 & 3 & 4 & 2 \\ 2 & 3 & 4 & 1 \\ 4 & 1 & 2 & 3 \\ 2 & 1 & 4 & 3 \\ 4 & 3 & 2 & 1 \end{pmatrix}.$$

Any ordering of three events can be found as subsequence of one row. If three particular events occur as subsequence of a row, we say that a row covers the three events. For example $(1, 2, 3)$ is a subsequence of the fourth row, $(1, 3, 2)$ is a subsequence of the second row, $(2, 3, 4)$ is covered by the third row, and so on and so forth.

SCAs are relevant for testing applications where the order of events is decisive. Examples of respective event sequences in such applications are user actions for user-interface testing, visited web pages in dynamic web applications, method calls for unit-testing in object-oriented programming, and shared

variable accesses in multi-threaded programs as we already mentioned. If an SCA of strength t is used as basis for a test plan for such applications, i.e., each row of the SCA is turned into the specification of a test run that imposes a particular order on relevant event, not all permutations of events will be tested in general, but at least we have the guarantee that the potential interaction of any t events will be tested at least once.

In practice, a direct application of SCAs for testing is often impaired by additional constraints on the order of events. It can be necessary to exclude, for example, that a “paste” event happens before a “copy” event when testing a user interface. Also, the conditions that identify the sequences that should be covered can vary and often involve quite complex definitions. For example, to test thread interleavings, one could require to test all sequences such that one variable is written by one thread and subsequently read by another thread such that there is no write operation between them [5], [6]. Hence, quite expressive constraints and variations from standard SCAs have to be taken into account. Furthermore, sometimes certain orderings are regarded as redundant and should be avoided to reduce testing costs. For example, the order in which devices are connected to a computer is not relevant if the computer is not booted.

One approach to address such considerations is to accordingly modify precomputed SCAs as exemplified by Kuhn et al. [4]. This means that any test sequence which, e.g., violates some ordering constraints has to be removed from the SCA. To maintain coverage, removed sequences have to be replaced by permutations thereof that comply to the problem specific requirements. This is not always possible in a straightforward way and can result in a considerable and in principle avoidable overhead regarding the size of arrays. On the other hand, developing and maintaining dedicated algorithms to compute variations of SCAs usually comes with high costs and is not preferable if requirements change over time—which is a daily aspect in real-world system development—or one wants to experiment with different designs.

We propose to use *answer-set programming* (ASP) [7]–[9] for computing SCAs and variations thereof. ASP is a genuine declarative programming paradigm where a problem is encoded by means of a logic program such that the solutions of a problem correspond to the models, called *answer sets*, of the program. On the one hand, as an expressive high-level specification language, it allows to state complex coverage criteria, involving constraints and complex, possibly recursive, definitions, in a concise and *elaboration-tolerant* way, i.e., small variations in a problem specification require only small modifications of the program representation. On the other hand, SCAs can be efficiently computed through highly optimised ASP solvers [10], [11]. Since it requires only little effort to state quite complex coverage conditions in ASP, a tester is able to rapidly specify

and to experiment with different versions of SCAs.

This paper is organised as follows. In Section II, we review SCAs and ASP. In Section III, we analyse the intrinsic problem complexity of SCA computation which indeed shows that ASP is a suitable computational means. Then, in Section IV, we show how SCAs can be generated using ASP. We present improved, sometimes optimal, upper bounds regarding the size of many SCAs. We furthermore present a greedy algorithm, based on ASP, for computing larger SCAs. In Section V, we turn towards a real-world example described by Kuhn et al. [4]. We discuss how the basic ASP encoding from Section IV can be refined, step-by-step, to take different constraints and problem variations into account. The resulting array is significantly smaller than the one of Kuhn et al. that was created by modifying a precomputed SCA. In fact, we show that our solution is optimal with respect to the specified coverage criteria. Finally, we discuss related work in Section VI and conclude in Section VII.

II. PRELIMINARIES

In this section, we review the formal definition of SCAs and give a brief background on ASP.

A. Sequence Covering Arrays (SCAs)

SCAs, introduced by Kuhn et al. [4], are combinatorial designs related to covering arrays. While covering arrays require that each t -way combination of parameters occurs at least once in a test case for some fixed t , SCAs take the order of events into account and require that each t -sequence of events is tested in at least one test sequence in that order, where a t -sequence over a set S of symbols is a sequence of t pairwise distinct elements of S . Following Kuhn et al. [4], we formally define SCAs as follows.

Definition 1: A *sequence covering array* (SCA) with parameters n , S , and t , or an (n, S, t) -SCA for short, is an $n \times |S|$ matrix M of symbols from a finite set S of symbols such that

- (i) each row of M is a permutation of S , and
- (ii) for each t -sequence $\sigma = (s_1, s_2, \dots, s_t)$ over S , there is at least one row $\varrho = (a_{i1}, \dots, a_{i|S|})$ in M such that σ is a subsequence of ϱ .

We say that an (n, S, t) -SCA is of *strength* t and of *size* n .

Definition 2: The *sequence covering array number* for S and t , $\text{SCAN}(S, t)$, is the smallest n such that an (n, S, t) -SCA exists.

An (n, S, t) -SCA is *optimal* if $\text{SCAN}(S, t) = n$. As usual, l is a *lower bound* for $\text{SCAN}(S, t)$ if $l \leq \text{SCAN}(S, t)$, and u is an *upper bound* for $\text{SCAN}(S, t)$ if $\text{SCAN}(S, t) \leq u$. We will also denote an $(n, \{1, \dots, s\}, t)$ -SCA as an (n, s, t) -SCA with $\text{SCAN}(s, t)$ for brevity.

For illustration, the following matrix M constitutes an optimal (7, 5, 3)-SCA:

$$M = \begin{pmatrix} 5 & 2 & 3 & 1 & 4 \\ 3 & 2 & 5 & 4 & 1 \\ 1 & 5 & 4 & 3 & 2 \\ 3 & 4 & 5 & 1 & 2 \\ 4 & 2 & 5 & 1 & 3 \\ 2 & 4 & 3 & 1 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}.$$

Each of the seven rows is a permutation of the set $S = \{1, \dots, 5\}$ and each 3-sequence over S is covered by at least one row. For instance, the 3-sequence (5, 3, 4) is covered by the first row of M , and (3, 4, 5) is covered by the fourth row of M (as well, (3, 4, 5) is covered by the last row of M). Note that there are $5 \cdot 4 \cdot 3 = 40$ such 3-sequences.

A collection of precomputed SCAs of strength 3 and 4 is available online [12]. These SCAs were computed using a simple greedy algorithm introduced by Kuhn et al. [4]. To compute a t -strength SCA for a set S of events, this algorithm iteratively computes single rows of the SCA: In each iteration, it computes a fixed number of permutations of S . Then, it selects the permutation π that obtains maximal coverage of previously uncovered t -sequences as the next row of the SCA. After that, π in reverse order, π' , is added. Adding π' is justified because π' always covers the same number of previously uncovered t -sequences as π [4]. This procedure is iterated until all t -sequences are covered.

Recently, this algorithm has been extended to deal with forbidden orderings of events. In particular, if event x must not occur before event y in any test case, it is possible to specify the pair (x, y) as additional input of the algorithm. Subsequently, only rows that do not contain (x, y) as subsequence are added. Also, if a constraint is specified, the heuristic to add rows in reverse order is disabled.

Though the greedy algorithm can take simple constraints into account, one downside is that more complex constraints or other variations from plain SCAs arising from the requirements of different test scenarios are hard to incorporate. To overcome this shortcoming, we use ASP in what follows as a declarative tool to compute SCAs and demonstrate that quite complex constraints can be incorporated into a solution in a concise and elaboration-tolerant way, and with ease.

B. Answer-Set Programming (ASP)

ASP [7]–[9] is a relatively new declarative programming paradigm. The underlying idea of ASP is to declaratively represent a computational problem as a logic program whose models, called “answer sets” [13], correspond to the solutions, and to find the answer sets for that program using an ASP solver. Due to the expressiveness of ASP, allowing, e.g., to represent aggregates and recursive definitions, and due to the continuous improvements of the efficiency of ASP solvers, such as `clasp` [14], we argue that ASP can efficiently and

effectively be used to compute SCAs. Indeed, ASP has been used in a wide range of applications from different fields, such as semantic-web reasoning [15], systems biology [16], planning [17], diagnosis [18], [19], configuration [20], multi-agent systems [21], cladistics [22], [23], game content generation [24], and superoptimisation [25]. For a comprehensive introduction to ASP, we refer to the textbook by Baral [9].

In what follows, we recapitulate the basic elements of ASP. An *answer-set program* is a finite set of rules of the form

$$a_0 :- a_1, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n, \quad (1)$$

where $n \geq m \geq 0$, a_0 (called the *head* of the rule) is a propositional atom or \perp , a_1, \dots, a_n are propositional atoms, and the symbol “not” denotes *default negation*. The sequence $a_1, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n$ comprises the *body* of the rule. If $a_0 = \perp$, then rule (1) is a *constraint* (in which case a_0 is usually omitted). The intuitive reading of a rule of form (1) is that whenever a_1, \dots, a_m are known to be true and there is no evidence for any of the default negated atoms a_{m+1}, \dots, a_n to be true, then a_0 has to be true as well. Note that \perp can never become true.

An *answer set* for a program is defined following Gelfond and Lifschitz [26]. An *interpretation* I is a finite set of propositional atoms. An atom a is *true* under I if $a \in I$, and *false* otherwise. A rule r of form (1) is true under I if $\{a_1, \dots, a_m\} \subseteq I$ and $\{a_{m+1}, \dots, a_n\} \cap I = \emptyset$ implies $a_0 \in I$. We say that I is a *model* of a program P if each rule $r \in P$ is true under I . Finally, I is an *answer set* of P if I is a subset-minimal model of P^I , where P^I is defined as the program that results from P by deleting all rules that contain a default negated atom from I , and deleting all default negated atoms from the remaining rules.

Programs can yield no answer set, one answer set, or many answer sets. For instance, the program

$$\begin{aligned} p &:- \text{not } q, \\ q &:- \text{not } p \end{aligned} \quad (2)$$

has two answer sets: $\{p\}$ and $\{q\}$.

When we represent a problem in ASP, some rules “generate” answer sets corresponding to “possible solutions”, and some “eliminate” the answer sets that do not correspond to solutions. The rules in program (2) are of the former kind; constraints are of the latter kind. For instance, adding the constraint

$$\perp :- p$$

to a program P eliminates all answer sets of P containing p . In particular, adding $\perp :- p$ to program (2) eliminates the answer set $\{p\}$.

When we represent a problem in ASP, we often use special constructs of the form $l\{a_1, \dots, a_k\}u$ (called *cardinality expressions*) where each a_i is an atom and l and u are nonnegative integers denoting the *lower bound* and the *upper bound* of the cardinality expression [27]. Programs using

these constructs can be viewed as abbreviations for particular normal programs [28]. Such an expression describes the subsets of the set $\{a_1, \dots, a_k\}$ whose cardinalities are at least l and at most u . In heads of rules, cardinality expressions generate answer sets containing subsets of $\{a_1, \dots, a_k\}$ whose cardinality is at least l and at most u . When used in constraints, they eliminate answer sets that contain such respective subsets.

A group of rules that follow a particular pattern can often be described in a compact way using *schematic variables*. For instance, we can write the program $p_i :- \text{not } p_{i+1}$, ($1 \leq i \leq 7$) as follows:

$$\begin{aligned} & \text{index}(1), \text{index}(2), \dots, \text{index}(7), \\ & p(i) :- \text{not } p(i+1), \text{index}(i). \end{aligned}$$

ASP solvers compute an answer set for a given program that contains variables after “grounding” the program, e.g., by the grounder `gringo` [29]. A grounder systematically replaces each rule r with variables by its ground instances that result from r by uniformly replacing each variable by constants from the program. Variables can also be used “locally” to describe a list of literals. For instance, the rule $1\{p_1, \dots, p_7\}1$ can be represented as $1\{p(i) : \text{index}(i)\}1$.

In addition to the constructs above, current state-of-the-art ASP solvers support many further language extensions like *functions*, *built-in arithmetics*, *comparison predicates*, *aggregate atoms*, *maximisation* and *minimisation statements*, as well as *weak constraints*.

In the remainder of this paper, we use the syntax that is supported by the solver `clasp` along with the grounding tool `gringo` when presenting programs [30]. Note that, at term positions, upper-case letters denote variables, while lower-case letters denote constant symbols.

For illustrating problem solving in ASP, consider the following encoding of the 3-colorability problem (3COL):

```
colour(red;green;blue).
1 {asgn(N,C) : colour(C)} 1 :- node(N).
:- edge(X,Y), asgn(X,C), asgn(Y,C).
```

The first rule abbreviates three facts that state that red, green, and blue are colours, respectively. The second rule is a choice rule. Its intuitive reading is that if N is a node, then both an upper bound and a lower bound on the number of colours assigned to this node, expressed by `asgn(N,C)`, is 1. This means that each node gets assigned precisely one colour from the set of available colours defined by `colour/1`. The last rule is a constraint that forbids that there is an edge between any two nodes with the same colour. If the above program is joined with facts over `edge/2` and `node/1` that represent a graph G , the answer sets correspond one-to-one to the valid 3-colourings of G .

Sometimes, one is not only interested in arbitrary solutions to a problem but in solutions that are optimal according to some preference relation. `clasp` supports *maximise* and

minimise statements that allow the express such preferences. For illustration, assume that, for some reason, we want to minimise the number of blue nodes in the above 3COL example. This can be expressed by simply adding the following *minimise* statement:

```
#minimize[asgn(N,blue) : node(N)].
```

The meaning of such a statement is that `clasp` computes answer sets where the sum of literals `asgn(N,blue)`, where N is a node, is minimal among all answer sets.

III. PRELUDE: COMPLEXITY OF SCA GENERATION

Deciding whether a logic program has an answer-set is NP-complete, thus computing answer-sets can be quite expensive. Indeed, the runtime of ASP solvers is exponential with respect to the number of atoms in the worst case. In this section, we analyse the computational worst-case complexity of generating SCAs. We assume that the reader is familiar with the basic concepts of complexity theory. For more information about complexity theory, we refer to the reference textbook by Papadimitriou [31].

For our complexity analysis, we actually study a slight generalisation of the problem of generating SCAs. On the one hand, usually not all permutations of events are allowed for testing, some could be excluded for various reasons. On the other hand, usually not all t -sequences need to be covered, some may be forbidden or regarded as redundant. We next formalise this natural generalisation as a decision problem and study its complexity.

Definition 3: An instance of the *generalised event sequence testing* (GEST) problem is a tuple (S, P, T, k) , where P be is a set of permutations of a set S of symbols, T is a set of t -sequences over S with $t \geq 2$, and k is a positive integer. A tuple (S, P, T, k) is a yes-instance of GEST iff there exists a matrix M with at most k rows such that

- (i) each row of M is an element from P , and
- (ii) for each t -sequence $\sigma = (s_1, s_2, \dots, s_t)$ from T , there is at least one row $\rho = (a_{i1}, \dots, a_{i|S|})$ in M such that σ is a subsequence of ρ .

Theorem 1: GEST is NP-complete.

Proof:

Membership. We first show that GEST is in NP. Any instance (S, P, T, k) of GEST can be decided by non-deterministically “guessing” a $k \times |S|$ matrix M of symbols from S and checking conditions (i) and (ii) from Definition 3 in polynomial time.

Hardness. To show NP-hardness, we reduce the NP-hard problem of checking set coverage to GEST. Formally, an instance of *set cover* (SC) is a tuple (V, F, k) , where V is a set of elements, F is a collection of subsets of V , and k is a positive integer. A tuple (V, F, k) is a yes-instance of SC iff there is a subcollection $F' \subseteq F$ of size at most k whose union contains each element of V . It is well known that SC is NP-complete [32].

Let (V, F, k) be an instance of SC. Assume that “ \square ” is a separation symbol not contained in V . Define

$$S = V \cup \{\square\}.$$

For each $f \in F$, construct a permutation π_f of S by arbitrarily arranging the symbols from f before \square and the symbols in $V \setminus f$ after \square . Define

$$P = \{\pi_f \mid f \in F\}$$

and

$$T = \{(v, \square) \mid v \in V\}.$$

Note that $|P| = |F|$. We show that (V, F, k) is a yes-instance of SC iff (S, P, T, k) is a yes-instance of GEST.

Assume that (V, F, k) is a yes-instance of SC. Hence, there exists a set $F' \subseteq F$ of size at most k whose union contains each element of V . Construct a matrix M such that $\pi_f \in P$ is a row of M iff $f \in F'$. Clearly, M is a matrix from symbols from S that satisfies condition (i) of Definition 3. We show that M satisfies condition (ii) as well. Towards a contradiction, assume that there is a 2-sequence (v, \square) in T and there is no row in M such that v occurs before \square . Since (V, F, k) is a yes-instance of SC, F' contains at least one set f with $v \in f$. Since π_f is a row of M , and v occurs before \square in π_f by construction, we arrive at a contradiction. Hence, (S, P, T, k) is a yes-instance of GEST.

For the converse, assume that (S, P, T, k) is a yes-instance of GEST. Hence, there exists a matrix M that satisfies conditions (i) and (ii) from Definition 3. We show that then (V, F, k) is a yes-instance of SC. Define set F' as a subset of F that contains an element $f \in F$ iff (*) there is a row π of M such that all elements of f occur before \square in π . Clearly, F' is of size at most k since M consists of at most k rows, and, for any row of M , precisely one $f \in F$ satisfies (*). It remains to show that for each $v \in V$, v is contained in some set in F' . Towards a contradiction, assume that for some $v \in V$ there is no set in F' that contains v . Since (S, P, T, k) is a yes-instance of GEST, and $(v, \square) \in T$, it follows that in one row π_f of M , v occurs before \square . By construction of F' , F' contains a set $f \in F$ consisting of all symbols of π_f that occur before \square . Hence, $v \in f$ which contradicts the assumption that no such set in F exists. So, (V, F, k) must be a yes-instance of SC. ■

Hence, any approach that is capable of deciding problems in GEST cannot avoid worst-case exponential runtime behaviour unless $P = NP$. Note that the SCA generation problems studied in this paper are instances of GEST. Moreover, for any problem in NP, there exists a uniform ASP encoding [33], [34]. Hence, NP-completeness of GEST further justifies ASP as a tool to formalise and compute such problems.

Although GEST is NP-complete, one could further ask whether GEST is *fixed-parameter tractable* for a suitable problem parameter. Roughly speaking, fixed-parameter

tractability means that a problem can be solved efficiently, i.e., in polynomial time, for fixed values of the parameter; details on parameterised complexity can be found elsewhere [35], [36]. A natural choice for such a parameter for a problem instance (S, P, T, k) would be the size k of the SCA because it can be assumed to be small in practice. We denote the parameterised version of GEST with k as parameter as the *standard parameterisation of GEST*.

In more formal terms, a parameterisation of a decision problem is obtained by assigning a natural number to each problem instance. A parameterised decision problem is fixed-parameter tractable if a problem instance x with parameter k can be decided in running time $f(k) \cdot |x|^{O(1)}$, where f is a computable function which is independent of $|x|$.

In standard complexity theory, problems are classified and ordered into hierarchies using polynomial-time reductions. Under parameterised complexity, parameterised reductions, so-called *fpt-reductions*, are used for this purpose. An fpt-reduction from a parameterised problem P to a parameterised problem Q is a function ϕ such that for any instance x of P

- (i) $\phi(x)$ can be computed in time $f(k) \cdot |x|^{O(1)}$, where k is the parameter of x ,
- (ii) $\phi(x)$ is a yes-instance of Q iff x is a yes-instance of P , and
- (iii) if k is the parameter of x and k' is the parameter of $\phi(x)$, then $k' \leq g(k)$, for some computable function g .

The class FPT contains all fixed-parameter tractable problems. Note that FPT is closed under fpt-reductions. Similar to the polynomial hierarchy in standard complexity theory, a hierarchy of classes $W[i]$ has been introduced with FPT at its lowest level. In particular, $FPT = W[0]$ and $W[i] \subseteq W[j]$, for all $i \leq j$. All classes $W[i]$ are closed under fpt-reductions. Moreover, analogous to $P \subseteq NP$, it is not known whether the inclusions $W[i] \subseteq W[j]$ are proper, but most experts believe this to be the case. The following result implies that GEST is not in FPT unless the W -hierarchy collapses up to the second level.

Theorem 2: The standard parameterisation of GEST is $W[2]$ -complete.

Proof:

Consider an instance (V, F, k) of SC. If we take k , i.e., the size of the subcollection $F' \subseteq F$ whose union contains each element of V , as parameter, SC is $W[2]$ -complete [35].

Membership. To show membership in $W[2]$, we use an fpt-reduction from GEST to SC. Let (S, P, T, k) be an instance of GEST. For any $\pi \in P$, construct a set $f_\pi \subseteq T$ that contains any t -sequence $\tau \in T$ iff the elements of τ occur in π in the same order as in τ . Define

$$F = \{f_\pi \mid \pi \in P\}.$$

We show that (S, P, T, k) is a yes-instance of GEST iff (T, F, k) is a yes-instance of SC.

Assume that (S, P, T, k) is a yes-instance of GEST. Hence, there exists a matrix M satisfying conditions (i)

and (ii) from Definition 3. Define F' as the subset of F that contains f_π iff π is a row of M . Clearly, the size of F' is at most k . It remains to show that the union of the elements of F' contain each $\tau \in T$. Towards a contradiction, assume that there is an element $\tau \in T$ such that no set in F' contains τ . Since (S, P, T, k) is a yes-instance of GEST, M contains a row $\pi \in P$ such that the symbols in τ occur in π in the same order as in τ . Thus $\tau \in f_\pi$. Since $f_\pi \in F'$, we arrive at a contradiction. Therefore, (T, F, k) must be a yes-instance of SC.

Assume now that (T, F, k) is a yes-instance of SC. Hence, there is a subset $F' \subseteq F$ of size at most k whose union contains each element of T . Construct a matrix M according to Definition 3 such that M contains, for each $f \in F'$, one row $\pi \in P$ that satisfies $f_\pi = f$. Clearly, M consists of at most k rows from P . It remains to show that M satisfies condition (ii) of Definition 3. Towards a contradiction, assume that there is a t -sequence $\tau \in T$ such that no row contains the symbols in τ in the same order as in τ . Since (T, F, k) is a yes-instance of SC, there is a set $f \in F'$ that contains τ . Since M contains a row π with $f_\pi = f$, it follows from the construction of f_π that π contains the symbols in τ in the same order as in τ . We thus arrive at a contradiction, and so (S, P, T, k) is yes-instance of GEST.

Hardness. To show that GEST is W[2]-hard, we define an fpt-reduction from SC to GEST. In fact, the reduction used in the hardness proof of Theorem 1 is such an fpt-reduction since the problem parameter k is preserved. Hence, W[2]-hardness of GEST follows. ■

Hence, even if we are interested only in relatively small test plans, it is presumably not possible to avoid worst-case exponential runtime.

IV. SCA COMPUTATION

We now discuss how ASP can be used to generate SCAs. Our goal is not only to present approaches to compute generic SCAs, i.e., SCAs created without additional constraints or requirements, rather we want to demonstrate that ASP can be used as an efficient and effective declarative tool to compute SCAs tailored to specific test scenarios. We in particular demonstrate that (i) the declarative nature of ASP encodings can help to state complex coverage criteria, involving constraints and possibly recursive definitions with ease in a concise and elaboration-tolerant way, and (ii) when the declarative nature of ASP encodings is coupled with ever-improving efficiency of ASP solvers, even simple encodings that closely reflect the problem statement in natural language can provide better SCAs (e.g., smaller SCAs) compared to those obtained from a dedicated algorithm.

Ahead of our discussion in Section V addressing how different problem elaborations can be incorporated into a single answer-set program, we introduce in what follows an answer-set program for computing generic SCAs. This program will serve as basis for further problem elaboration

```

% ASP encoding for (n,s,3)-SCAs
sym(1..s). row(1..n).

% guess happens-before relation
1{hb(N,X,Y),hb(N,Y,X)}1 :- row(N),
                               sym(X;Y), X != Y.

% happens-before is transitive
hb(N,X,Z) :- hb(N,X,Y), hb(N,Y,Z).

% happens-before is irreflexive
:- hb(N,X,X).

% check if each 3-sequence is covered
covered(X,Y,Z) :- hb(N,X,Y), hb(N,Y,Z).
:- not covered(X,Y,Z),
   sym(X;Y;Z), X!=Y, Y!=Z, X!=Z.

```

Figure 1. ASP encoding $\Pi^3(n, s)$.

discussed in the sequel. We also introduce a new greedy approach that combines a simple variation of the basic ASP encoding with an iterative greedy procedure.

A. Basic Encoding

To begin with, we present an ASP program for computing (n, s, t) -SCAs with $t = 3$. We assume throughout that $s \geq 3$. Note that this program can be changed in a straightforward way to obtain encodings for any fixed $t > 3$. An encoding for SCAs where t is not fixed can be obtained using *disjunctive ASP* [13]—this is, however, beyond the scope of this article. For the sake of understandability, we introduce our encoding step-by-step.

1) *Encoding:* We start by expressing that the symbols of the array are integers between 1 and s , and row indices of the SCA correspond to integers 1 to n . Note that s and n function as parameters of the program:

```
sym(1..s). row(1..n).
```

For the representation of the SCA, we use the predicate $hb(N, X, Y)$ expressing that in row N event symbol X happens before symbol Y . The basic idea is that we will define this happens-before relation in a way that it is, for each row, a strict total order on the event symbols.

The first rule states that for any two distinct symbols X and Y in each row, either X happens before Y or Y happens before X :

```
1 {hb(N,X,Y),hb(N,Y,X)} 1 :- row(N),
                               sym(X;Y), X != Y.
```

We need further rules to guarantee that the happens-before relation is indeed a strict total order. In particular, we need rules that guarantee that the happens-before relation is transitive and irreflexive. Now, transitivity can be expressed in a straightforward way:

```
hb(N,X,Z) :- hb(N,X,Y), hb(N,Y,Z).
```

Directly expressing inductive definitions as above is a particular strength of ASP and distinguishes it from related declarative approaches that are more based on the semantics of classical first-order logic.

To state that the happens-before relation is irreflexive, a simple additional constraint is required.

$$\text{:- hb}(N, X, X) .$$

Hence, it is forbidden that a symbol occurs before itself.

This finally implies that the happens-before relation is a strict total order on the event symbols $\{1, \dots, s\}$ which further implies that each row is a permutation of the event symbols when we order them according to the happens-before relation.

It only remains to require that each 3-sequence of symbols is covered by some row. Observe that a 3-sequence is a triple of pairwise distinct symbols. A 3-sequence (X, Y, Z) is covered if X happens before Y and Y happens before Z in some row N . We finally define covered 3-sequences and forbid that a 3-sequence is not covered:

$$\begin{aligned} \text{covered}(X, Y, Z) & \text{ :- hb}(N, X, Y), \text{ hb}(N, Y, Z) . \\ \text{:- not covered}(X, Y, Z), \\ & \text{sym}(X; Y; Z), X \neq Y, Y \neq Z, X \neq Z. \end{aligned}$$

The entire ASP program $\Pi^3(n, s)$ with parameters n and s for generating $(n, s, 3)$ -SCAs is given in Figure 1.

Intuitively, each answer set of program $\Pi^3(n, s)$ represents an $(n, s, 3)$ -SCA. In fact, the answer sets of $\Pi^3(n, s)$ and the $(n, s, 3)$ -SCAs are in a one-to-one correspondence. This relation can be formalised as follows:

Definition 4: An answer set X of $\Pi^3(n, s)$, for $s \geq 3$, represents an $n \times s$ matrix M iff, for any $i, j, 1 \leq i < j \leq s$, and any $r, 1 \leq r \leq n$, $M_{r,i} = s_1$ and $M_{r,j} = s_2$ precisely in case X contains the atom $\text{hb}(r, s_1, s_2)$.

Proposition 1: Each answer set of $\Pi^3(n, s)$ represents a single $(n, s, 3)$ -SCA, and each $(n, s, 3)$ -SCA is represented by a single answer set of $\Pi^3(n, s)$.

For illustration, to compute a $(7, 5, 3)$ -SCA, `gringo` and `clasp` can be invoked as follows:

```
gringo sca-3.gr -c n=7,s=5 | clasp.
```

File `sca-3.gr` contains program $\Pi^3(n, s)$. The `gringo` option `-c n=7,s=5` instantiates the program parameters n and s to 7 and 5, respectively. Any resulting answer set corresponds to a $(7, 5, 3)$ -SCA. For instance, in some answer set, the first row of the SCA M given in Section II-A is encoded by the atoms

$$\begin{aligned} & \text{hb}(1, 1, 4), \text{ hb}(1, 3, 1), \text{ hb}(1, 2, 3), \\ & \text{hb}(1, 5, 2), \text{ hb}(1, 5, 3), \text{ hb}(1, 2, 1), \\ & \text{hb}(1, 3, 4), \text{ hb}(1, 2, 4), \text{ hb}(1, 5, 4), \\ & \text{hb}(1, 5, 1). \end{aligned}$$

To compute more than one $(7, 5, 3)$ -SCA, an upper bound on the number of answer sets that `clasp` should compute can

Table I
UPPER BOUNDS n FOR $\text{SCAN}(s, 3)$ OBTAINED BY KUHN ET AL. [4] AND OUR ASP ENCODING. A STAR INDICATES AN OPTIMAL BOUND.

s	n (Kuhn et al. [4])	n (ASP)
5	8	7*
6	10	8*
7	12	8*
8	12	8*
9	14	9*
10	14	9*
11	14	10
12	16	10
13	16	10
14	16	10
15	18	10
16	18	10
17	20	11
18	20	12
19	22	12
20	22	12
21	22	12
22	22	12
23	24	13
24	24	13
25	24	14
26	24	14
27	26	14
28	26	14
29	26	14
30	26	15
40	32	17
50	34	18
60	38	20
70	40	22
80	42	23

be specified as an integer option (0 means that all answer sets are computed).

2) *Discussion:* Program $\Pi^3(n, s)$ nicely illustrates how challenging search problems can be concisely encoded using ASP: The program consists of only seven rules that closely reflect the problem statement in natural language. We note that only little training time is needed to enable a tester to use ASP for test authoring. This is mainly because of the genuine declarative nature of ASP, which does not require specialised knowledge on data structures or algorithms. A more experienced ASP user needs about 15 minutes to develop a program such as the one given in Figure 1.

Also, by using our ASP encoding $\Pi^3(n, s)$ and the ASP solver `clasp`, we could improve known upper bounds for many SCAs significantly. A comparison of the SCAs generated using ASP and the greedy algorithm of Kuhn et al. [4] is given in Table I. Computation times for the reported upper bounds range from fractions of a second to 180 minutes. We have considered strength 3 SCAs for five to 80 events. The known upper bounds reported by Kuhn et al. [4] could be improved throughout. The more events are considered, the more drastic are the improvements; for some arrays, we need up to 46.88% less test sequences. Such savings are especially significant in settings where running single test sequences are costly.

For small SCAs—viz. for 5 to 10 events—the new upper bounds are actually optimal bounds. Optimality of upper bounds was established using ASP itself. To show that an (n, s, t) -SCA is optimal, we try to compute an $(n - 1, s, t)$ -SCA. If this fails, i.e., the ASP solver terminates without returning an answer set, the (n, s, t) -SCA is indeed optimal. Since $\text{SCAN}(10, 3) = 9$, 9 is a trivial lower bound for any $\text{SCAN}(s, 3)$ with $s > 10$. Note that greedy algorithms, or any approaches based on incomplete search, are unable to prove optimal bounds or to establish lower bounds at all.

A limitation of using the ASP encoding $\Pi^3(n, s)$ concerns scalability. Although memory usage is always limited by a polynomial with respect to the input parameters n and s , the runtime of `clasp` is worst-case exponential for encoding $\Pi^3(n, s)$. On the other hand, the greedy approach of Kuhn et al. [4] seems to scale quite well; the authors report on SCAs for up to 80 events not only for strength 3 arrays, but they also consider arrays of strength 4 where our ASP approach quickly reaches its limits.

B. Greedy Algorithm

In the remainder of this section, we introduce and discuss an ASP-based greedy algorithm, inspired by that of Kuhn et al. [4], for computing larger SCAs. The motivation to study such an algorithm is to combine the modelling capabilities of ASP, especially in the light of constraints and problem elaborations (as detailed in the next section), with the scalability of a greedy approach.

In this context, we also mention that the greedy algorithm of Kuhn et al. has a certain weakness, which is related to the heuristic that for any newly computed sequence the reverse sequence is added as well (cf. Section II). As we will show next, this makes the algorithm inherently unable to compute optimal SCAs in general. Actually, the inability to find optimal SCAs follows immediately from the observation that some optimal SCAs, e.g., $(7, 5, 3)$ -SCAs, are of odd size. However, ASP can be used to show that even optimal SCAs of even size cannot be found by that greedy approach in general. The idea is to augment program $\Pi^3(n, s)$ by a rule that states that every second row is the inversion of the previous one. This is simply expressed by the following rule:

$$\text{hb}(N, X, Y) \text{ :- row}(N), \text{hb}(N-1, Y, X), \\ N \# \text{mod } 2 == 0.$$

Here, predicate `#mod` is the usual modulo operation. Hence, the intuitive reading of this rule is that for any row with even index N , the happens-before relation is the inverse of the happens-before relation of the preceding row $N-1$. We know already from Table I that any $(8, 6, 3)$ -SCA is optimal. However, $\Pi^3(8, 6)$ augmented by the above rule yields no answer set, which shows that $(8, 6, 3)$ -SCAs cannot be computed by the greedy algorithm of Kuhn et al. [4]. Next, we present an ASP-based greedy algorithm inspired by that of Kuhn et al. that does not rely on adding inverted rows.

Require: s is the number of symbols.
Ensure: N represents an $(n, s, 3)$ -SCA.
1: $N \leftarrow \emptyset$
2: $n \leftarrow 0$
3: **repeat**
4: $n \leftarrow n + 1$
5: $X \leftarrow$ answer set of $\Pi^3_{\text{grdy}}(s, n) \cup N$
6: $N \leftarrow N \cup X|_{\text{hb}/3}$
7: **until** N represents an $(n, s, 3)$ -SCA

Figure 2. Greedy algorithm for computing an $(n, s, 3)$ -SCA.

1) *Encoding:* Figure 2 represents our ASP-based greedy algorithm for computing SCAs. The main idea is to compute one row of a SCA at a time instead of computing the entire array. In each iteration, one further row is computed using ASP where the number of covered 3-sequences is maximised. For this purpose, we use program $\Pi^3_{\text{grdy}}(s, i)$, which is depicted in Figure 3. Program $\Pi^3_{\text{grdy}}(s, i)$ takes the number s of events and a row index i as parameters. Both the ASP encoding and the greedy algorithm are introduced only for SCAs of strength 3. However, versions for computing SCAs of strength greater than 3 are obtained in a straightforward way. To obtain a program for strength 4 SCAs, for example, only the last two rules of $\Pi^3_{\text{grdy}}(s, i)$ have to be replaced by the following two rules:

```
covered(W, X, Y, Z) :- hb(n, W, X), hb(n, X, Y),
                        hb(n, Y, Z).
#maximize[covered( _, _, _, _ )].
```

Program $\Pi^3_{\text{grdy}}(s, i)$ is quite similar to $\Pi^3(n, s)$. However, each answer set of $\Pi^3_{\text{grdy}}(s, i)$ corresponds only to a single row with index i of an SCA. The idea is to represent preceding rows with index 1 to $i - 1$ by means of facts `hb/3`. These facts are joined with $\Pi^3_{\text{grdy}}(s, i)$. Then, the answer sets of $\Pi^3_{\text{grdy}}(s, i)$ correspond to those rows that obtain maximal coverage of previously uncovered 3-sequences. The encoding follows the *guess, check, and optimise pattern*, i.e., we use guessing rules to span the search space, constraints to filter unwanted solution candidates, and rules that express a preference relation on answer sets. In particular, rule

```
#maximize[covered( _, _, _ )].
```

states that we seek for answer sets with a maximal number of covered 3-sequences.

The algorithm itself is rather simple (cf. Figure 2): It takes parameter s as input and computes an $(n, s, 3)$ -SCA. Initially, the set N that represents a (partial) SCA by means of facts `hb/3` equals the empty set. In each iteration, $\Pi^3_{\text{grdy}}(s, i) \cup N$ is used to compute the next row of the SCA that obtains maximal increase of previously uncovered 3-sequences. The respective `hb/3` facts for that row are then added to N . This procedure iterates until no uncovered 3-sequences are left (the ASP solver itself will indicate that no further optimisation is possible). Since the computation of optimal answer sets can become very time consuming, we additionally impose an

```

% guess single row with index i of an
% (n,s,3)-SCA
sym(1..s).

% guess happens-before relation
1{hb(i,X,Y),hb(i,Y,X)}1 :- sym(X;Y), X != Y.

% happens-before is transitive
hb(i,X,Z) :- hb(i,X,Y), hb(i,Y,Z).

% happens-before is irreflexive
:- hb(i,X,X).

% maximise coverage
covered(X,Y,Z) :- hb(N,X,Y), hb(N,Y,Z).
#maximize[covered(_,_,_)].

```

Figure 3. ASP encoding $\Pi_{grdy}^3(s, i)$.

upper bound on the time that is spent for optimising answer sets, thus improvements in each step will not be maximal in general. However, this seems to be a reasonable compromise regarding runtime and the size of computed SCAs. We used time limits of up to 10 minutes for computing single rows, depending on the problem size.

2) *Discussion:* To sum up our results so far, our analysis of the heuristic proposed by Kuhn et al. [4] using ASP has pinpointed some shortcomings of the former and has helped us to learn more about the problem at hand. Furthermore, we have proposed a new greedy algorithm making use of a slight variation of $\Pi^3(n, s)$. The ASP solver takes care entirely of the greedy optimisation of the single rows of the SCA. The algorithm thus only keeps track of the partial SCA and incrementally calls the ASP solver to compute new rows.

Table II summarises a comparison of our greedy ASP algorithm with the greedy algorithm of Kuhn et al. [4] for strength 3 and 4 SCAs involving 10 to 80 events. For strength 3 SCAs, our algorithm is competitive with that of Kuhn et al. and upper bounds could be improved throughout by some rows. For strength 4 SCAs, the greedy ASP approach is feasible for up to 40 symbols where upper bounds could be improved even more drastically than for strength 3 SCAs. However, we were not able to compute SCAs for 40 to 80 symbols, which shows a limitation of our ASP-based approach that is probably acceptable unless the need for larger instances with a high level of interaction is indeed motivated by some application scenario. This limitation basically comes from the huge number of 4-sequences that need to be covered and that are represented by the program. Here, it is to mention that scalability is certainly a characteristic strength of the simple greedy algorithm of Kuhn et al., since dedicated data structures, e.g., efficient bit-vectors, can be used for representing covered sequences. However, by using ASP we get bounds for strength 3 SCAs for up to 80 symbols and can also improve bounds for strength 4 SCAs for up to 40 symbols. Again, we emphasise that our goal is not

Table II
COMPARISON OF OUR GREEDY ASP APPROACH AND THAT OF KUHN ET AL. [4]: UPPER BOUNDS n FOR SCAN($s, 3$) AND SCAN($s, 4$).

s	$t = 3$		$t = 4$	
	Kuhn et al. [4]	ASP	Kuhn et al. [4]	ASP
10	14	11	66	55
20	22	17	120	104
30	26	22	156	149
40	32	26	182	181
50	34	29	204	-
60	38	32	222	-
70	40	35	238	-
80	42	36	250	-

to compute generic SCAs but to allow a tester to express different requirements with little effort, by adding or changing some rules of the ASP program, which can readily be done using the greedy ASP approach. We pursue this issue in the next section.

V. PROBLEM ELABORATIONS

Next, we turn to the actual strengths of using ASP as an elaboration tolerant representation formalism for event sequence testing. We describe how ASP can be used for generating SCAs in a scenario that involves additional constraints and other problem variations that make it impossible to directly use precomputed SCAs. In particular, we use a *real-world testing problem* described by Kuhn et al. [4] for making our point. The specification of this testing problem is as follows: There are five different devices that have to be connected to a laptop. These devices can be connected before or after a boot-up phase. Further actions that have to be performed on the laptop are opening an application and initiating a scanning process. The peripherals can be connected to the laptop in any order; however, the order of events influences the functionality of the system. Thus, SCAs lend themselves as a basis for a suitable testing plan.

There are eight events relevant for testing: connecting devices ($p1, \dots, p5$), booting the system ($boot$), starting an application ($appl$), and running a scan ($scan$). Testing in this scenario is rather time consuming since it requires setting up the system manually. Therefore, obtaining an optimal test plan is a clear desideratum. Following Kuhn et al., only SCAs of strength 3 are considered to keep the size of the test plan reasonable.

A. Forbidden Sequences

For eight events, optimal SCAs of strength 3 comprise eight rows. However, we cannot use precomputed $(8, 8, 3)$ -SCAs since certain constraints regarding the order of events have to be taken into account. While most events can happen in any order, starting the application cannot happen before the system is booted, and running a scan requires that the application is already running.

1) *Encoding*: Instead of covering all 3-sequences, we want to generate SCAs such that (i) in each row, `boot` happens before `appl` and `appl` happens before `scan`, and (ii) all 3-sequences such that `boot` happens before `appl` and `appl` happens before `scan` are covered by at least one row. We only have to slightly modify program $\Pi^3(n, s)$ to account for (i) and (ii). First, instead of integers to denote events, we would like to use more descriptive constant symbols. Thus, we replace `sym(1..s)` in $\Pi^3(n, s)$ by

```
sym(boot;p1;p2;p3;p4;p5;appl;scan) .
```

Concerning (i), we define which orderings are excluded and add a respective constraint that forbids that event *a* happens before *b* if “*a* before *b*” is excluded.

```
excluded(scan,appl) .
excluded(appl,boot) .
excluded(X,Z) :- excluded(X,Y) ,
                  excluded(Y,Z) .
:- hb(_,X,Y) , excluded(X,Y) .
```

Regarding (ii), we simply define those 3-sequences that are not consistent with the excluded orderings as already covered:

```
covered(X,Y,Z) :- excluded(X,Y) ,
                  sym(X;Y;Z) .
covered(X,Y,Z) :- excluded(X,Z) ,
                  sym(X;Y;Z) .
covered(X,Y,Z) :- excluded(Y,Z) ,
                  sym(X;Y;Z) .
```

We denote the resulting program as $\Pi_1^3(n)$.

2) *Discussion*: Recall that (8, 8, 3)-SCAs are optimal for eight symbols. Since, $\Pi_1^3(8)$ does not yield any answer set, it follows that the stipulation on admissible orderings requires additional rows. In this case, this is because the number of 3-sequences that can be covered by a single row is reduced if certain events are required to happen in a strict order. Indeed, a solution for $\Pi_1^3(9)$ can be computed, hence 9 is an optimal bound for an SCA satisfying that each row is consistent with the specified ordering constraints. The solver `clasp` needs fractions of a second to find an SCA of size 9 and about 1 minute for checking optimality.

B. Redundant Sequences

Besides forbidden orderings, we also have to deal with redundant sequences: If devices are connected to the laptop before the boot-up phase, the order is not relevant. In fact, we only require strength 3 coverage for events `p1`, ..., `p5`, `appl`, and `scan`. Concerning the interaction of events `p1`, ..., `p5`, and `boot`, we regard strength 2 coverage as sufficient, i.e., we are only interested in whether the connection of the peripherals happens before or after the boot-up phase. Hence, we need a variable strength SCA, in which we seek to have strength 2 coverage for one set of events and strength 3 coverage for another one.

1) *Encoding*: First, we add two sets of facts to declare the sets of events for which we want to obtain strength 2 and strength 3 coverage, respectively:

```
threeWay(p1;p2;p3;p4;p5;appl;scan) .
twoWay(boot;p1;p2;p3;p4;p5) .
```

Next, we have to modify some rules where appropriate. In particular, we only want to cover 3-sequences over symbols from `threeWay/1`. Hence, we rewrite rule

```
threeSeq(X,Y,Z) :- sym(X;Y;Z) , X!=Y, Y!=Z,
                  X!=Z.
```

into

```
threeSeq(X,Y,Z) :- threeWay(X;Y;Z) ,
                  X!=Y, Y!=Z, X!=Z.
```

To address two-way coverage of the symbols from predicate `twoWay/1`, we add two further rules:

```
covered(X,Y) :- hb(_,X,Y) .
:- twoWay(X;Y) , X != Y, not covered(X,Y) .
```

The resulting program is denoted by $\Pi_2^3(n)$.

2) *Discussion*: Program $\Pi_2^3(n)$ incorporates both forbidden configurations and redundant sequences. Respective SCAs can be obtained for $n = 8$ already. SCAs of size 8 are indeed optimal arrays, which follows from the observation that $\Pi_2^3(7)$ yields no answer set at all. It takes on average 0.1 seconds to compute the first answer set of a size 8 SCA when using `clasp` as ASP solver. Showing optimality, i.e., that no size 7 SCA exists, needs several minutes.

The solution approach of Kuhn et al. uses a precomputed (12, 7, 3)-SCA to account for the seven events `p1`, ..., `p5`, `scan`, and `appl`. In a post-processing step, rows that are not consistent with the ordering constraints (cf. Section V-A) are replaced. However, this requires that further rows are added to preserve coverage. Note that this testing application was considered before the greedy algorithm was extended to directly express simple constraints [4]. In a further manual post-processing step, to account for the two-way coverage with respect to events `p1`, ..., `p5`, and `boot`, Kuhn et al. add `boot` as the first event of each row. Finally, an additional row is added, in which all events `p1`, ..., `p5` are arranged prior to `boot`, thereby obtaining strength 2 coverage between `boot` and events `p1`, ..., `p5`. The resulting array consists of 19 rows.

The first thing to note is that using ASP enabled us to easily embed the additional requirements directly in the ASP program rather than employing an ad hoc and mostly manual approach. Furthermore, using ASP significantly reduced the size of the resulting SCA by eleven rows (57.94%), cf. Table III.

C. Adding Attributes to Events

The next problem elaboration that we consider is related to the way the peripherals are connected to the laptop. Devices

Table III
TEST PLAN OF SIZE 8 FOR THE LAPTOP APPLICATION OBTAINED FROM AN ANSWER SET OF $\Pi_4^3(8)$.

row	event 1	event 2	event 3	event 4	event 5	event 6	event 7	event 8
1	p3(l)	p2(r)	p1(b)	p4	boot	appl	scan	p5
2	boot	p4	p1(r)	appl	p5	p3(l)	scan	p2(b)
3	boot	appl	scan	p1(r)	p2(b)	p4	p3(l)	p5
4	p1(r)	p2(b)	p5	p3(l)	boot	appl	scan	p4
5	boot	p3(b)	p5	p1(r)	appl	p4	p2(l)	scan
6	p4	boot	p2(b)	p5	appl	p1(l)	scan	p3(r)
7	boot	appl	scan	p5	p3(l)	p4	p2(b)	p1(r)
8	p5	boot	p2(l)	p4	p3(r)	appl	scan	p1(b)

p1, p2, and p3 have to be connected to USB ports. Three ports are available: left, right, and back. In each test sequence, one port has to be assigned to a USB device.

1) *Encoding*: Predicate `port(N, X, Y)` states that USB device X is connected to port Y in row N of the array. This assignment should satisfy the following coverage criteria:

- (i) each USB device has to be connected to each port at least once, and
- (ii) connections to the ports after the boot event should be made in any possible order.

The above requirements can be formalised using few further rules.

In the following rules, we first specify the USB ports and devices. Then, it is expressed that each USB device is assigned to precisely one port in each test sequence. Finally, USB devices must not be connected to the same port in any sequence.

```
usbPort(right; left; back).
usbDevice(p1; p2; p3).
1{port(N, X, Y) : usbPort(Y) } 1 :- row(N),
                                usbDevice(X).
:- port(N, X, Y), port(N, Z, Y), X != Z.
```

Next, we state coverage criterion (i):

```
portCov(X, Y) :- port(N, X, Y).
:- usbDevice(X), usbPort(Y),
   not portCov(X, Y).
```

Lastly, we add rules for coverage criterion (ii):

```
portSeq(X, Y, Z) :- usbPort(X; Y; Z),
                    X != Y, X != Z, Y != Z.
seqCov(N, X, Y, Z) :- hb(N, boot, X),
                      hb(N, X, Y),
                      hb(N, Y, Z).
pSeqCov(R, S, T) :- seqCov(N, X, Y, Z),
                    port(N, X, R),
                    port(N, Y, S),
                    port(N, Z, T).
:- portSeq(X, Y, Z), not pSeqCov(X, Y, Z).
```

Let us denote the resulting program by $\Pi_3^3(n)$.

2) *Discussion*: Note that the additional conditions regarding the USB ports do not result in larger SCAs, still SCAs

of size 8 can be obtained by computing the answer sets of $\Pi_3^3(8)$. Clearly, 8 is also an optimal bound. The runtime of the ASP solver is not affected by the additional requirements.

Kuhn et al. deal with the issue of USB ports by adding respective port assignments in a post-processing step once an SCA is computed. However, they do not provide details on which basis this is done, i.e., it is not clear if or in what sense they strive for systematic coverage.

D. Expressing Preferences

Any answer set of $\Pi_3^3(n)$ represents one admissible test plan for the application under test. Although each such SCA satisfies all of the requirements discussed so far, different SCAs could differ in their fault detection potential.

We next augment program $\Pi_3^3(n)$ by rules that state a preference relation among solutions, similar to program $\Pi_{grady}^3(\cdot, \cdot)$ from the previous section. In particular, although any SCA guarantees full three-way interaction coverage for some specified events, the degree of four-way coverage of events may differ from one SCA to another. We will use the number of covered 4-sequences as discrimination criterion regarding the quality of solutions and consequently prefer SCAs that cover more 4-sequences over SCAs that cover fewer.

1) *Encoding*: We define program $\Pi_4^3(n)$ as $\Pi_3^3(n)$ augmented by the following rules:

```
covered(W, X, Y, Z) :- hb(N, W, X), hb(N, X, Y),
                       hb(N, Y, Z).
#maximize[covered(_, _, _, _)].
```

The first rule defines which 4-sequences are covered, the second rule states that the number of covered 4-sequences should be maximised. The complete ASP encoding $\Pi_4^3(8)$ is given in Figure 4.

2) *Discussion*: An SCA of size 8 corresponding to an answer set of $\Pi_4^3(8)$ is given in Table III. In the computation of the SCA, `clasp` has been configured to optimise a solution until no improvements can be found for 15 minutes.

On the other hand, Kuhn et al. [4] have not handled preferences over solutions at all. The algorithm of Kuhn et al. is tailored for computing a single SCA. Thus, it may be hard to use such an algorithm to directly deal with optimisation issues, since this requires that solutions should be efficiently enumerated.

This case study demonstrates that often generic SCAs cannot be used in a real world scenario without significant modifications. In general, such modifications lead to a considerable overhead or are not feasible at all. By using ASP, however, a test author has a tool to state different requirements relevant for individual scenarios. Often, this will need only little effort such as adding few rules.

VI. RELATED WORK

The ASP-based approach introduced in this paper is the first account of an approach for directly generating SCAs in the presence of expressible constraints and problem elaborations. We note, however, that after the previous conference version of this paper [1], our idea was picked up soon by Banbara et al. [37]. They proposed a constraint-programming encoding called the *incidence-matrix model* for generating SCAs. Although we could either reproduce or improve all bounds for $\text{SCAN}(s, 3)$ reported by Banbara et al. [37] in this paper, the encoding based on the incidence-matrix model scales better than ours for $\text{SCAN}(s, 4)$ SCAs.

Closely related to our work are techniques for computing covering arrays (CAs), which we will review next. An overview of different approaches and tools for generating CAs is given by Grindal, Offutt, and Andler [2]. There, greedy algorithms that construct one row at a time are quite common. The most prominent representative is the AETG system [38]. Our greedy approach to compute SCAs is close in spirit to AETG-like algorithms since it also proceeds row by row. Also, meta-heuristics, like simulated annealing, tabu search, or genetic algorithms, have been applied for constructing CAs [39], [40] (cf. respective overview articles for more details [2], [3]). Greedy algorithms usually scale well while meta-heuristics tend to produce arrays of smaller sizes [39]. However, neither greedy techniques nor meta-heuristics can guarantee optimal bounds.

As a complete method being able to establish optimality of arrays, different SAT encodings have been considered [41], [42]. Similar to our ASP encoding, SAT encodings allow to compute combinatorial designs as a whole. From a computational point of view, SAT and ASP are closely related and ASP solvers like `clasp` use many techniques also used by SAT solvers like conflict-driven clause learning. In fact, `clasp` can be used as a SAT solver itself—it even outperformed state-of-the-art SAT solvers at the SAT 2011 competition. A distinctive feature of ASP compared to SAT is the high-level modelling capabilities of ASP that allow to model problems concisely at the first-order level as demonstrated by our SCA encodings. SAT is certainly a promising approach for tackling problems described in Section IV, i.e., for computing SCAs and checking optimality of upper bounds. However, the problem variations discussed in Section V require a formalism that allows for elaboration-tolerant representations, which is not a characteristic feature of SAT. Regarding modelling, it is to mention that Hnich et

```
% ASP encoding for the laptop example
sym(boot; p1; p2; p3; p4; p5; appl; scan).
row(1..n).

threeWay(p1; p2; p3; p4; p5; appl; scan).
twoWay(boot; p1; p2; p3; p4; p5).

% guess happens-before relation
1{hb(N, X, Y), hb(N, Y, X)}1 :- row(N),
                               sym(X;Y), X != Y.

% happens-before is transitive
hb(N, X, Z) :- hb(N, X, Y), hb(N, Y, Z).
% happens-before is irreflexive
:- hb(N, X, X).

% check three-way and two-way coverage
covered(X, Y, Z) :- hb(N, X, Y), hb(N, Y, Z).
:- not covered(X, Y, Z),
   threeWay(X;Y;Z), X!=Y, Y!=Z, X!=Z.
covered(X, Y) :- hb(_, X, Y).
:- twoWay(X;Y), X != Y, not covered(X, Y).

% excluded orderings
excluded(scan, appl).
excluded(appl, boot).
excluded(X, Z) :- excluded(X, Y), excluded(Y, Z).
:- hb(_, X, Y), excluded(X, Y).
covered(X, Y, Z) :- excluded(X, Y), sym(X;Y;Z).
covered(X, Y, Z) :- excluded(X, Z), sym(X;Y;Z).
covered(X, Y, Z) :- excluded(Y, Z), sym(X;Y;Z).

% coverage of USB ports
usbPort(right; left; back).
usbDevice(p1; p2; p3).
1{port(N, X, Y) : usbPort(Y)}1 :- row(N),
                               usbDevice(X).
:- port(N, X, Y), port(N, Z, Y), X != Z.

portCov(X, Y) :- port(N, X, Y).
:- usbDevice(X), usbPort(Y), not portCov(X, Y).

portSeq(X, Y, Z) :- usbPort(X;Y;Z),
                   X!=Y, X!=Z, Y!=Z.
seqCov(N, X, Y, Z) :- hb(N, boot, X), hb(N, X, Y),
                    hb(N, Y, Z).
pSeqCov(R, S, T) :- seqCov(N, X, Y, Z),
                   port(N, X, R), port(N, Y, S), port(N, Z, T).
:- portSeq(X, Y, Z), not pSeqCov(X, Y, Z).

% maximise covered 4-sequences
covered(W, X, Y, Z) :- hb(N, W, X), hb(N, X, Y),
                      hb(N, Y, Z).
#maximize[covered(_,_,_,_)].
```

Figure 4. ASP encoding $\Pi_4^3(8)$.

al. [41] and Banbara et al. [42] initially considered constraint programming (CP) models, which are subsequently translated to SAT. Although this has not been considered, further constraints, at least forbidden tuples, could be incorporated rather easily into the CP model. A comparison of ASP and constraint (logic) programming (CLP) is given in a related

article [43]. There, the authors conclude that ASP allows for more declarative and concise problem representations and is easier to learn for beginners than CLP.

The need for stating constraints and other user requirements in combinatorial interaction testing for real-world applications has been discussed by different authors [38], [44]–[50]. The prevalent approach is to first generate a CA and then to delete and permute rows that are not consistent with certain requirements. The number of rows that need to be replaced can be vast and this approach can lead to a considerable increase of the array size [49]. This applies not only for CAs but for SCAs as well as we have illustrated in the previous section. Another common method requires remodelling of the specification [38], [45].

The tool PICT [47], also based on an AETG-like greedy algorithm, allows to directly express constraints; however, the details how this is realised are not accessible.

Cohen, Dwyer, and Shi [48], [49] introduced approaches that integrate techniques for generating covering arrays with SAT to deal with constraints. Forbidden tuples are represented as Boolean formulas and a SAT solver is used to compute models. They integrated SAT with greedy AETG-style algorithms and also with simulated annealing. Hence, their approach is closely related to our integration of ASP into a greedy procedure. Calvagna and Gargantini [50] follow a similar approach but they use an SMT solver instead of a SAT solver, which offers a richer language than plain SAT solvers. In their approach, constraints are stated as formal predicate expressions. Besides SMT, Calvagna and Gargantini also considered a model checker for verifying test predicates which would also be suitable for specifications involving temporal constraints and state transitions.

Bryce and Colbourn [46] distinguish forbidden tuples and tuples that should be avoided. They refer to the latter as soft constraints and they present an algorithm for generating CAs that avoids the violation of soft constraints. However, their algorithm cannot guarantee that certain tuples are avoided, hence it cannot deal with forbidden tuples or other hard constraints. Using ASP, soft constraints can be easily expressed by means of minimise or maximise statements. Some ASP solvers, like DLV [51], allow to express soft constraints even more directly (in the case of DLV, in the form of *weak constraints*). We illustrated in the previous section how one can combine hard integrity constraints with soft constraints to express that uncovered 4-sequences should be avoided. For even more fine-grained modelling, ASP allows to assign different priorities to soft constraints and maximise, resp., minimise, statements.

VII. CONCLUSION AND FUTURE WORK

In this paper, we dealt with the generation of SCAs, which have recently been advocated as suitable combinatorial design concepts for event sequence testing [4]. In particular, we applied ASP as a declarative approach for generating

SCAs. While the only previously introduced algorithm is an AETG-like greedy algorithm [4], ASP can be used as an exact method that combines high-level modelling capabilities involving recursive definitions, default negation, hard constraints, soft constraints, and aggregates with highly performative search engines [10], [11].

Our contributions can be summarised as follows:

- We established new complexity results related to computing SCAs. In particular, we showed that GEST is NP-complete and its standard parameterisation is $W[2]$ -complete.
- We introduced a novel technique to use ASP for computing SCAs as a whole. The SCAs obtained using ASP are significantly smaller than those generated using the greedy algorithm of Kuhn et al. [4]. For some SCAs, optimality of upper bounds could be established.
- We integrated our ASP approach into a greedy algorithm that allows to compute SCAs in a one-row-at-a-time fashion. Hence, we obtain a more scalable algorithm without sacrificing the modelling capacities of ASP for specifying complex testing problems.
- We dealt with problem elaborations that are indispensable for testing real-world applications. In particular, we addressed how constraints and other application-specific requirements can be handled directly at the level of the ASP representation without a further need for post-processing steps.

To summarise, our contribution is two-fold: On the one hand, we introduced and showed feasibility of a new approach for generating SCAs that can be readily used as it is. On the other hand, we regard this work as a contribution towards methodology. While ASP is well established in other communities as a method to address problems from the area of artificial intelligence and knowledge representation, there is too little awareness of ASP in the software-engineering community. Hence, we want to promote ASP as an approach to tackle challenging problems in the realm of combinatorial testing. Besides improving the state-of-the-art of event sequence testing, our aim is to show that ASP provides a tool that enables a tester to rapidly specify problems and to experiment with different formulations at a purely declarative level. ASP solvers are then used for computing solutions without the need of post-processing steps or developing dedicated algorithms.

For future work, we plan to deal with versions of SCAs for different testing applications like testing of concurrent programs where the order of shared variable accesses was identified as crucial for triggering certain bugs that are otherwise hard to evoke [6], [52]. We want to address not only the problem of statically generating suitable designs, but we also want to do this in an *online fashion* where an ASP solver is coupled with a scheduler to improve coverage with respect to different interleaving metrics. Such an online approach

would also allow to deal with, e.g., exceptional events in a more interactive testing environment. In the long term, we plan to develop support for a tester regarding modelling of a system's test space without requiring expert knowledge on ASP—a front-end language for ASP tailored to specific testing domains could be the right way of doing this.

ACKNOWLEDGMENT

This work was supported by the Austrian Science Fund (FWF) under project P21698. We also would like to thank D. Richard Kuhn for providing us with related work [4].

REFERENCES

- [1] E. Erdem, K. Inoue, J. Oetsch, J. Pührer, H. Tompits, and C. Yilmaz, "Answer-set programming as a new approach to event-sequence testing," in *Proceedings of the 3rd International Conference on Advances in System Testing and Validation Lifecycle (VALID 2011)*. XPS, 2011, pp. 25–34.
- [2] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: A survey," *Software Testing, Verification & Reliability*, vol. 15, no. 3, pp. 167–199, 2005.
- [3] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys*, vol. 43, no. 2, pp. 11:1–11:29, 2011.
- [4] D. R. Kuhn, J. M. Higdon, J. Lawrence, R. Kacker, and Y. Lei, "Combinatorial methods for event sequence testing," in *Proceedings of the 5th International Conference on Software Testing, Verification and Validation (ICST 2012)*. IEEE, 2012, pp. 601–609.
- [5] M. J. Harrold and B. A. Malloy, "Data flow testing of parallelized code," in *Proceedings of the 8th International Conference on Software Maintenance (ICSM 1992)*. IEEE Computer Society Press, 1992, pp. 272–281.
- [6] S. Lu, W. Jiang, and Y. Zhou, "A study of interleaving coverage criteria," in *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2007, pp. 533–536.
- [7] V. Marek and M. Truszczyński, "Stable models and an alternative logic programming paradigm," in *The Logic Programming Paradigm: A 25-Year Perspective*. Springer, 1999, pp. 375–398.
- [8] I. Niemelä, "Logic programs with stable model semantics as a constraint programming paradigm," *Annals of Mathematics and Artificial Intelligence*, vol. 25, no. 3-4, pp. 241–273, 1999.
- [9] C. Baral, *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, 2003.
- [10] M. Denecker, J. Vennekens, S. Bond, M. Gebser, and M. Truszczyński, "The second answer set programming competition," in *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009)*, ser. Lecture Notes in Computer Science, vol. 5753. Springer, 2009, pp. 637–654.
- [11] F. Calimeri, G. Ianni, and F. Ricca, "The third open answer set programming competition," *CoRR*, vol. abs/1206.3111, 2012. [Online]. Available: <http://arxiv.org/abs/1206.3111>
- [12] "Combinatorial testing for event sequences," http://csrc.nist.gov/groups/SNS/acts/sequence_cov_arrays.html, last visited: July 11, 2012.
- [13] M. Gelfond and V. Lifschitz, "Classical negation in logic programs and disjunctive databases," *New Generation Computing*, vol. 9, pp. 365–385, 1991.
- [14] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "Conflict-driven answer set solving," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*. AAAI Press/MIT Press, 2007, pp. 386–392.
- [15] A. Polleres, "Semantic Web Languages and Semantic Web Services as Application Areas for Answer Set Programming," in *Nonmonotonic Reasoning, Answer Set Programming and Constraints*, 2005.
- [16] S. Grell, T. Schaub, and J. Selbig, "Modelling biological networks by action languages via answer set programming," in *Proceedings of the 22nd International Conference on Logic Programming (ICLP 2006)*, ser. Lecture Notes in Computer Science, vol. 4079. Springer, 2006, pp. 285–299.
- [17] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres, "Planning under incomplete knowledge," in *Proceedings of the First International Conference on Computational Logic (CL 2000)*, ser. Lecture Notes in Computer Science, vol. 1861. Springer, 2000, pp. 807–821.
- [18] T. Eiter, W. Faber, N. Leone, and G. Pfeifer, "The diagnosis frontend of the DLV system," *AI Communications*, vol. 12, no. 1-2, pp. 99–111, 1999.
- [19] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry, "An A-prolog decision support system for the Space Shuttle," in *Proceedings of the 3rd International Symposium on Practical Aspects of Declarative Languages (PADL 2001)*, ser. Lecture Notes in Computer Science, vol. 1990. Springer, 2001, pp. 169–183.
- [20] T. Soinen and I. Niemelä, "Developing a declarative rule language for applications in product configuration," in *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL 1999)*, ser. Lecture Notes in Computer Science. Springer, 1999.
- [21] C. Baral and M. Gelfond, "Reasoning agents in dynamic domains," in *Logic-based Artificial Intelligence*. Kluwer Academic Publishers, 2000, pp. 257–279.
- [22] E. Erdem, V. Lifschitz, and D. Ringe, "Temporal phylogenetic networks and logic programming," *Theory and Practice of Logic Programming*, vol. 6, no. 5, pp. 539–558, 2006.
- [23] D. R. Brooks, E. Erdem, S. T. Erdogan, J. W. Minett, and D. Ringe, "Inferring phylogenetic trees using answer set programming," *Journal of Automated Reasoning*, vol. 39, no. 4, pp. 471–511, 2007.

- [24] A. M. Smith and M. Mateas, "Answer set programming for procedural content generation: A design space approach," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 187–200, 2011.
- [25] M. Brain, T. Crick, M. De Vos, and J. Fitch, "TOAST: Applying answer set programming to superoptimisation," in *Proceedings of the 22nd International Conference on Logic Programming (ICLP 2006)*, ser. Lecture Notes in Computer Science. Springer, 2006.
- [26] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *Proceedings of the 5th Logic Programming Symposium*, MIT Press, 1988, pp. 1070–1080.
- [27] P. Simons, I. Niemelä, and T. Soinen, "Extending and implementing the stable model semantics," *Artificial Intelligence*, vol. 138, no. 1–2, pp. 181–234, 2002.
- [28] P. Ferraris and V. Lifschitz, "Mathematical foundations of answer set programming," in *We Will Show Them! Essays in Honour of Dov Gabbay, Volume One*. College Publications, 2005, pp. 615–664.
- [29] M. Gebser, T. Schaub, and S. Thiele, "Gringo: A new grounder for answer set programming," in *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007)*, ser. Lecture Notes in Computer Science, vol. 4483. Springer, 2007, pp. 266–271.
- [30] "Potassco—the Potsdam answer set solving collection," <http://potassco.sourceforge.net>, last visited: July 11, 2012.
- [31] C. H. Papadimitriou, *Computational Complexity*. Addison-Wesley, New York, 1994.
- [32] R. Karp, "Reducibility among Combinatorial Problems," in *Complexity of Computer Computations*. Plenum Press, 1972, pp. 85–103.
- [33] J. S. Schlipf, "The expressive powers of the logic programming semantics," *Journal of Computer and System Sciences*, vol. 51, no. 1, pp. 64–86, 1995.
- [34] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, "Complexity and expressive power of logic programming," *ACM Computing Surveys*, vol. 33, no. 3, pp. 374–425, 2001.
- [35] R. Downey and M. R. Fellows, *Parameterized complexity*. New York: Springer, 1999.
- [36] J. Flum and M. Grohe, *Parameterized Complexity Theory*, ser. Text in Theoretical Computer Science. Springer, 2006.
- [37] M. Banbara, N. Tamura, and K. Inoue, "Generating event-sequence test cases by answer set programming with the incidence matrix," in *Technical Communications of the 28th International Conference on Logic Programming (ICLP 2012), LIPIcs, Schloss Dagstuhl*, vol. 12, 2012, pp. 86–97.
- [38] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," *IEEE Transactions on Software Engineering*, vol. 23, no. 7, pp. 437–444, 1997.
- [39] M. B. Cohen, P. B. Gibbons, and W. B. Mugridge, "Constructing test suites for interaction testing," in *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, 2003, pp. 38–48.
- [40] K. J. Nurmela, "Upper bounds for covering arrays by tabu search," *Discrete Applied Mathematics*, vol. 138, no. 1–2, pp. 143–152, 2004.
- [41] B. Hnich, S. D. Prestwich, E. Selensky, and B. M. Smith, "Constraint models for the covering test problem," *Constraints*, vol. 11, no. 2–3, pp. 199–219, 2006.
- [42] M. Banbara, H. Matsunaka, N. Tamura, and K. Inoue, "Generating combinatorial test cases by efficient SAT encodings suitable for CDCL SAT solvers," in *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2010)*, ser. Lecture Notes in Computer Science, vol. 6397. Springer, 2010, pp. 112–126.
- [43] A. Dovier, A. Formisano, and E. Pontelli, "An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 21, no. 2, pp. 79–121, 2009.
- [44] A. Hartman and L. Raskin, "Problems and algorithms for covering arrays," *Discrete Mathematics*, vol. 284, no. 1–3, pp. 149–156, 2004.
- [45] C. M. Lott, A. Jain, and S. R. Dalal, "Modeling requirements for combinatorial software testing," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–7, 2005.
- [46] R. C. Bryce and C. J. Colbourn, "Prioritized interaction testing for pair-wise coverage with seeding and constraints," *Information & Software Technology*, vol. 48, no. 10, pp. 960–970, 2006.
- [47] J. Czerwonka, "Pairwise testing in real world," in *Proceedings of the 24th Pacific Northwest Software Quality Conference (PNSQC 2006)*, 2006, pp. 419–430.
- [48] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints," in *Proceedings of the 16th ACM/SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2007, pp. 129–139.
- [49] —, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 633–650, 2008.
- [50] A. Calvagna and A. Gargantini, "A formal logic approach to constrained combinatorial testing," *Journal of Automated Reasoning*, vol. 45, no. 4, pp. 331–358, 2010.
- [51] "DLV system," <http://www.dlvsystem.com>, last visited: July 11, 2012.
- [52] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2008, pp. 329–339.

Soft Constraints in Feature Models: An Experimental Assessment

Jorge Barreiros^{1,2}

¹Instituto Superior de Engenharia de Coimbra
Instituto Politécnico de Coimbra,
Coimbra, Portugal
jmsousa@isec.pt

Ana Moreira²

²Dept. de Engenharia Informática
Universidade Nova de Lisboa
Caparica, Portugal
amm@fct.unl.pt

Abstract—Feature Models specify admissible configurations of products in Software Product Lines. Constraints are used to represent domain specific knowledge, such as requiring or excluding a feature in the presence of another. Configurations failing to conform to these constraints are deemed invalid. However, in many cases useful domain information cannot be expressed comfortably with such forceful, hard constraints. To overcome this problem, softer constraints, of less forcing nature, can be used. We describe a framework for including soft constraints in feature models based on propositional logic. Analysis procedures for detecting inconsistencies and conflicts in this framework are also described. Test sets are built by injecting soft constraints into publicly available feature models, recreating typical patterns of use. These features are then subjected to automated analysis to assess the prevalence of soft constraint related conflicts and interactions.

Keywords - Feature Models; Software Product Lines; Soft Constraints; Feature Consistency; Feature Interaction; Semantic Validation

I. INTRODUCTION

Soft constraints, described in [1], can be used to represent uncertain configuration information into feature models [2]. Feature models are frequently used in Software Product Line (SPL) development [3] for specifying valid product configurations, that is, configurations corresponding to a variant that can be created by an application engineer using the SPL. Product variants belonging to the same family are created by specifying a feature configuration, which is then realized by the composition of corresponding artifacts from a common pool of assets (such as requirements documents, design models, code, etc.).

Feature models identify valid configurations by using a feature tree annotated with additional domain constraints. These can be represented graphically (e.g., linking dependent features with a dependency arrow) or textually, by means of arbitrary cross-tree expressions (Boolean expressions depending on the configuration variables). Over-constraining may result in an inconsistent feature model, that is, one where no configuration exists, where all the constraints can be satisfied simultaneously.

Feature models can be represented using logic expressions according to well-known transformations

described in [4], [5]. A feature model expression is obtained by conjoining the feature tree expression with the domain constraints.

An example of a feature model can be found in Fig. 1, where *Sound*, *Keyboard*, and *Screen* are mandatory subfeatures of the root feature node *Phone*, while *MP3Player* and *Camera* are optional subfeatures. *Polyphonic* and *Monophonic* are mandatory and alternative subfeatures of the *Sound* feature, and *Monochromatic* and *Color* are alternative subfeatures of the *Screen* feature. One domain constraint is represented: the *requires* arrow describes that selection of the *Camera* feature implies the selection of the *Color* feature.

Links such as the one connecting *Camera* and *Color* in Fig. 1 describe *hard constraints*. Any configuration that does not respect this constraint is invalid. It can be the case, however, that domain information is not comfortably representable using such strict constructs. For example, a situation can be considered where the overwhelming majority of configurations do indeed respect a certain restriction, but a few exceptions may exist. In this case, restrictions on admissible configurations cannot be as strict. A simple example will be the case of a default selection for a group of alternative selections: if the parent feature of such group is selected, then the preferred alternative configurations may be suggested.

In [1], the use of soft constraints is proposed, similar to hard constraints but of less forcing nature, in these situations. The concept of soft constraint has been described earlier in the context of probabilistic feature models [6]. Probabilistic feature models extend standard feature models by the addition of “soft” constraints that are associated with a degree of probability. These are often obtained as the result of a feature mining processes. We consider the use of a similar concept in standard, deterministic feature models, avoiding the need to resort to mechanisms such as data mining or Bayesian networks to fully specify the required feature joint-probability distributions. The use of soft constraints allows richer semantics to be represented in feature models, with advantages such as enhanced analysis and improved configuration support. An example of such a constraint in Fig. 1 would be “*Sound suggests Polyphonic*”, expressing domain knowledge that indicates the more common sound configuration option. Naturally, soft constraints do not need to be restricted to parent-child features as described:

other relations such as “*Monophonic suggests Monochromatic*” can be represented. This type of constraints can be useful for efficiently capturing useful domain information that might be lost otherwise, as it is usually absent in standard feature models. It can be used to good effect for multiple purposes, depending on the specific semantics that are adopted as described later, such as allowing interactive configuration tools to suggest configuration choices to the user.

Using soft constraints also allows some semantic consistency analysis that would otherwise be impossible, e.g., if a suggested dependency can never be realized in a feature model, then probably something is not right. Suggestions may also be unsatisfiable for a certain valid partial configuration (e.g., suggestions cannot be satisfied simultaneously if one feature is selected), highlighting that a trade-off analysis may be in order.

We extend the work presented in [1] by:

- Describing structural patterns of application of soft constraints.
- Describing a process for injecting soft constraints into a feature model for the purpose of automated testing.
- Presenting an enhanced discussion of the impact of soft constraints in feature configuration and corresponding analysis technique.
- Using a prototype tool to collect and analyze experimental results using data from publicly available feature model repositories.

In Section II, we present motivating examples for our work. In Section III, we provide a detailed discussion of soft constraints, discussing benefits of their use, proposing a categorization of the different types of soft constraints, and discussing automated analysis procedures. Section IV presents some typical patterns of use of soft constraints, while in Section V the soft constraint injection algorithm is presented. The tool and experimental results are presented in Section VI, followed by a presentation of related work in Section VII and conclusions in Section VIII.

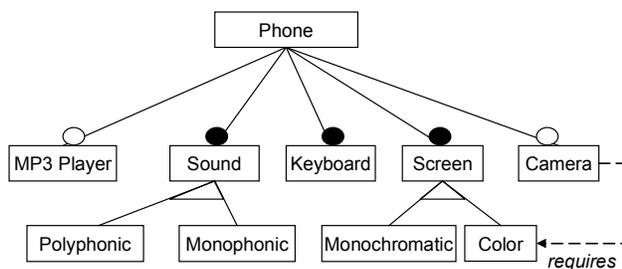


Figure 1. Mobile phone feature model.

II. MOTIVATION

Consider the example in Fig. 2, adapted from [6], where a feature model is used to describe configuration variability for an automobile vehicle. In this case, hard domain restrictions are used to enforce the selection of manual transmission in sports vehicles and to make sure that emission control techniques are always used in products destined for markets with stricter environmental legislations. While observance of such constraints is always found in valid products, soft constraints are used to represent relevant relations between features that, while not as critical or universally applicable as the hard constraints, are also important. In this case, it is well known that the USA market tends to favor vehicles with automatic transmission over those with manual transmission, while the converse is true for the European market. Using soft constraints, such information can be readily represented in the feature diagram, bringing in additional semantics that can be used to good effect.

Another example of the use of soft constraints can be found in Fig. 3. In this case, the feature model is used to represent dynamic variability of the runtime behavior of a real-time system. The system should adapt its behavior to conform to variations in its environment. The state of the operation environment is assessed by appropriate sensors and the corresponding features are (de)selected accordingly, with corresponding impact on the runtime behavior as dictated by the constraints. A base control task is to be active at all times, while fan control is only suggested if the temperature is medium, but mandatory if it reaches a high level. A filtering task is suggested if electric noise is detected.

The need to use soft constraints to describe the variability in this scenario is supported by the fact that the suggested (non mandatory) features may not always be selected because of limited resources (e.g., available CPU load). This means that a feature such as *Fan Control* may in fact remain unselected in the presence of its suggestor (i.e., the *Noisy* feature), which cannot be comfortably expressed using only hard constraints.

These examples suggest that soft constraints can be used to good effect in feature models, by allowing the inclusion of important domain information of non-forcing nature.

III. SOFT CONSTRAINTS

In this section, we discuss the benefits gained by using soft constraints in feature models and present a categorization of alternative semantics. We then discuss automated analysis procedures for identifying inconsistencies and other relevant information, such as features that cannot be selected if satisfaction of a soft constraint is sought.

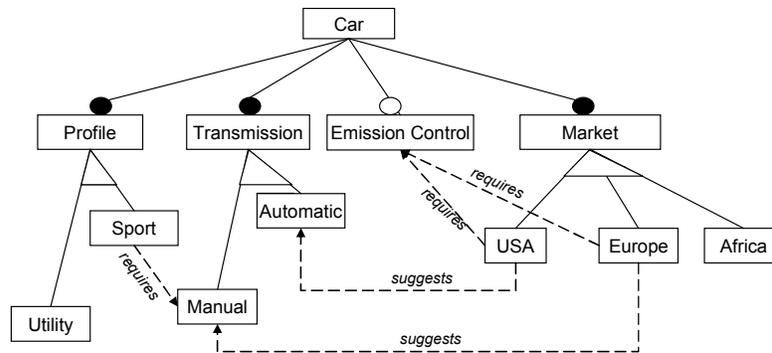


Figure 2. Feature model for car configuration

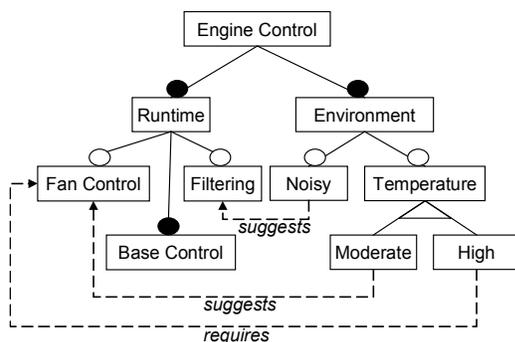


Figure 3. Engine control system

A. Benefits of soft constraints

Benefits of using soft constraints in feature models include:

- Improved configuration support:** Interactive configuration and completion techniques can assist the configuration of feature models by assessing the liveness of features after each configuration step. Starting from an empty configuration where all features are considered to be unspecified (neither selected or deselected), after a feature is selected or deselected by the user, the liveness of all features is re-evaluated with respect to the partial configuration already defined. Features that are found to be dead (always unselected) in that partial configuration can be safely deselected automatically. Conversely, features that are common to all configurations that include the partial configuration so far specified can be automatically selected. For example, if the developer specifies feature *C* in Fig. 4 to be selected, then features *D* and *E* can be automatically deselected by the configuration tool, as no valid configuration including feature *C* will contain either (i.e., both are dead in all configurations where *C* is selected). Similarly, *A* and *root* are common to all such configurations, so they can be selected automatically, leaving only feature *B* unspecified. Interactive configuration and completion tools can

use soft constraint information to make configuration suggestions to the user. For example, if “*A* suggests *B*”, the configuration tool can propose the selection of *B* by default whenever *A* is selected and *B* is unspecified. Also, if a valid configuration fails to conform to a large percentage of soft constraints, it can be flagged to the developer as suspicious. Feature configuration support for feature models with soft constraints is described in [7].

- Improved semantic-oriented consistency checks:** Standard consistency analysis of feature models is concerned with ensuring that valid configurations do exist. If soft constraints are present, it is possible to make sure that configurations are available that verify the suggested dependencies. If that is not the case, this may be a sign that an analysis or modeling error has occurred. For example, if it were actually impossible to configure a car for the European market with manual transmission despite such association being suggested (e.g., because of the unintended collateral side effect of some hard constraints), this would be highly suspicious and should be reported to the developer for additional consideration. This could be the case if hard domain restrictions would make it impossible to select a configuration where both such features are selected.
- Controlled generalization of feature models:** A generalization of a feature model is a transformation that increases the number of admissible configurations, making sure that previously valid configurations remain valid. In some cases, soft constraints can be used as a mechanism for controlled generalization of feature models. For example, if it was found, after creating the feature model in Fig. 2, that it should actually be possible, under certain circumstances, to produce vehicles without emission control for the USA market, the hard restriction that forbids such products from being created could be transformed into an equivalent soft constraint. This would have the benefit of preserving important domain information while accommodating the need to allow for spurious “rogue” configurations.

B. Semantics and Categorization

Soft constraints can be interpreted according to different semantics, from configuration suggestions (e.g., describing a predominant configuration options such as described in [6]) to stricter impositions that must be enforced if possible (i.e., a feature must be selected if possible). According to the adopted interpretation, different types of analysis and interpretations may be possible. Therefore, we must consider the possible semantics. These can be broadly categorized in two different categories:

- **Annotational:** A soft constraint with an annotational semantics does not impose any additional restriction when added to a feature model. Its main purpose is to embed domain information in the feature model to assist the configuration automation and semantic consistency checking. The validity of any specific product configuration is never influenced by the presence of an annotational soft constraint.
- **Normative:** A normative soft constraint must be considered when assessing the validity of a product configuration. These constraints represent configuration information that may potentially condition the validity of some configurations. A normative soft constraint must be satisfied if possible, but can be ignored otherwise. The concept of “possible satisfaction” is, generally, always dependent on the characteristics of the feature model and is also potentially dependent on domain-specific information (external to what is represented on the feature model: see below). A normative soft constraint may change the validity of a configuration (with respect to the unconstrained feature model), but it may never cause a feature model to become inconsistent. Normative constraints can be interpreted informally as meaning “requires-if-possible”, “may-require”, “require-if-does-not-make-configuration-invalid” or some other similar designation. Applying normative constraints entails the need to assess the “possibility” of selecting a specific feature. The topology of the feature model and cross-tree-constraints is always a decisive factor in making that assessment (i.e., it cannot be reasonably considered “possible” to select a feature when doing so would generate an invalid configuration). However, it may be the case that the feature model information is not sufficient to assess the possibility of selecting a feature: in this case, external factors, not represented in the feature model would come into play. This suggests the following additional characterization of normative constraints:

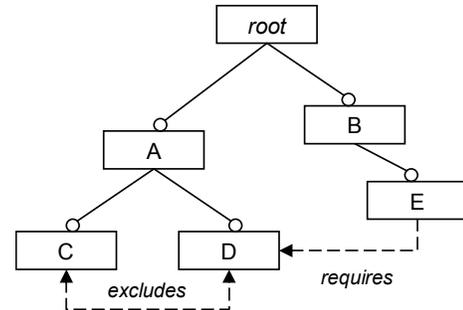


Figure 4. Iterative configuration example

- **Internal:** The feature model holds all the information required to assess selection possibility.
- **External:** The information in the feature model alone is not sufficient for assessing possibility of selection. External factors come into play.

In the example of Fig. 2, if the soft constraints are interpreted under annotational semantics, then any configuration that upholds the hard constraints is considered valid, regardless of complying or not with the soft constraints. On the other hand, if an (internal) normative semantic is considered, the following interpretation holds: “If the *USA* feature is selected, then the *Automatic* feature must be selected, unless doing so would generate an invalid configuration”. That is, a normative soft constraint should be interpreted as a hard constraint, unless doing so would turn an otherwise valid configuration into invalid. In Fig. 3, a potential example of external normative soft constraints is represented: in this case, the *Fan Control* feature should always be selected if the *Moderate* heat feature is selected, unless that is not possible, according to domain information that is not necessarily integrated in the feature model. For example, knowing that the implementations of the *Base Control*, *Fan Control*, and *Filtering* features compete for a limited resource (CPU load), assessing of the possibility of including the *Fan Control* feature must be conducted with respect to external information. It is out of the scope of this work to discuss how such external information would be obtained or retrieved – as examples, an oracle could be used to provide the required information or a domain specific ontology could be queried. External normative constraints require considering information beyond the one available on the feature model and will not be discussed further in this work. Therefore, in the remainder, when referring to normative soft constraints, internal semantics are assumed.

Table I presents a description of soft constraints and their intended meaning. Table II presents a summary overview of hard and soft constraint categorization and their effects on feature model consistency and configuration validity.

TABLE I. SOFT CONSTRAINT DESCRIPTION

Soft Constraint	Interpretation
A suggests B	$A \Rightarrow B$
A discourages B	$A \Rightarrow \neg B$
A absence-suggests B	$\neg A \Rightarrow B$
A absence-discourages B	$\neg A \Rightarrow \neg B$

TABLE II. CONSTRAINT CATEGORIZATION SUMMARY

Nature	Subtype	Affects FM consistency?	Affects config Validity?
Hard		Yes	Yes
Soft	Normative	No	Yes
	Annotational	No	No

C. Normative Soft Constraint Analysis

Normative soft constraints may change the assessment of the validity of configurations with respect to the unconstrained feature model. This results in a change of the model expression when a new soft constraint is introduced in an existing feature model. The effect of inserting an internal normative soft constraint (A suggests B) into a feature model with feature expression $F(A, B, \dots)$ can be obtained by considering that:

- Any configuration valid in the constrained model will also be valid in the unconstrained model. That is, $F(A, B, \dots)$ should hold.
- The soft constraint should be upheld ($A \Rightarrow B$), but not at the cost of invalidating the configuration. Knowing that the soft constraint fails to hold when A is selected and B is not, this will only be acceptable if switching the value of B , in this scenario, would not be allowed according to the unconstrained model¹, that is, $\neg F(A, \neg B, \dots)$.

These considerations lead to the following formulation:

$$F_S(A, B, \dots) = F(A, B, \dots) \wedge ((A \Rightarrow B) \vee \neg F(A, \neg B, \dots)) \quad (1)$$

where F_S is the resulting feature model expression.

Standard feature model techniques can be applied to analyze the resulting feature expression, e.g., satisfiability-based techniques are commonly applied to the analysis of feature model expressions [8], for tasks such as finding dead features. This can be also done in a feature model annotated with soft constraints by considering the relevant F_S .

¹ Strictly speaking, it would also be possible to satisfy the constraint by deselecting A rather than selecting B , but we find that solution to be counter-intuitive, with respect to the compositional approach inherent in the feature selection process. In other words, a constraint may force the selection of a feature the user has not selected, but will not force the deselection of a previously selected feature.

Equation (1) can be applied iteratively with respect to all soft constraints, in some priority order, to obtain the feature expression corresponding to a feature model with multiple soft constraints. Nevertheless, one difficulty must be pointed out. If F is in clause normal form (CNF), the standard input format for most SAT solvers, then the number of clauses will increase exponentially as additional normative soft constraints are composed. This makes it much more challenging to analyze normative soft constraints rather than their annotational counterparts. Fortunately, annotational soft constraints include valuable information that can be more efficiently subjected to automated analysis.

D. Soft Constraint Analysis

The inclusion of soft constraints in a feature model brings additional semantics that allow improved consistency and sanity checks to be performed. Annotational soft constraints do not alter in any way the space of admissible configurations. Nevertheless, the question of whether or not the soft constraints themselves can be upheld is relevant. In the remaining text, we assume an annotational interpretation of soft constraints.

When introducing a soft constraint into a feature model, all configurations previously valid will remain so. However, if the soft constraint impacts the feature model meaningfully, at least some of those configurations will fail to hold all the soft constraints (or else the soft constraint will be reproducing information already present in the feature model: an example would be a suggestion of inclusion of a parent feature). If no configuration exists where all the soft constraints are upheld, the soft constraints are inconsistent, in the sense that their simultaneous satisfaction is impossible (this is not the same as feature model inconsistency, as defined in the introduction).

An analysis procedure may be used to identify such situations. We begin by defining a constraint as *active* if its implicant (e.g. A in $A \Rightarrow B$) is true according to the expressions defined in Table II. It may be impossible to simultaneously activate all constraints according to the feature model. In this case, the constraint set is *orthogonal*. In this case, the constraints may not be satisfied simultaneously, because its implicants cannot be verified simultaneously. A more interesting situation occurs when all constraints can be active simultaneously, but satisfaction of the soft constraints is not possible. In this case, the soft constraints are said properly *inconsistent*.

Inconsistencies and orthogonally can be analyzed by verifying the satisfiability of Boolean propositions composed from the feature expression and soft constraint expressions. Although verifying the satisfiability of a proposition is an NP-Complete problem, SAT solvers have proven to be efficient tools for the majority of expression of practical interest for feature modeling [8]. Let \mathbf{F} be the feature expression, \mathbf{E} the conjunction of the soft constraint expressions and \mathbf{P} the conjunction of the soft constraint implicants according to the expressions found in Table I:

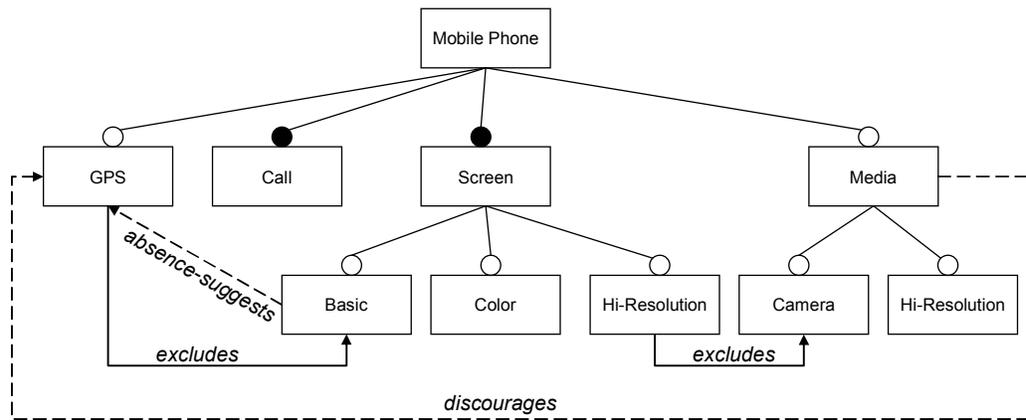


Figure 5. Feature model annotated with soft constraints (source: adapted from www.splot-research.org)

1. Check for satisfiability of the conjunction $F P$. If it is unsatisfiable, then the soft constraints are inconsistent due to orthogonality.
2. A proper inconsistency in a non-orthogonal set of constraints can be identified by assessing the satisfiability of $F P E$. If that expression is found to be unsatisfiable, then an inconsistency is detected.

Inconsistencies are not the only interesting interaction between soft constraints and feature models. In fact, any constraint of the format presented in Table II may be satisfied by falsifying the implicant. It may be the case where a soft constraint may only be satisfied by this recourse. In this case, we say the implicant is *hidden* by the soft constraint, in the sense that it may never be made true if the soft constraint is to be upheld. This is relevant, as it makes it possible to identify specific configuration profiles that must be upheld for satisfaction of the soft constraint to be possible. In the sequel, we will refer to hidden implicants as hidden features, although strictly speaking the implicant may not correspond to the selection of a single feature.

Hidden features can be identified in the following way. Let F be the feature expression, E the conjunction of the soft constraint expressions, and I_c the implicant of soft constraint c . Then:

1. If $F E$ is satisfiable, then at least one configuration exists that satisfies the feature model and soft constraints, and proceed to step 2.
2. For all I_c : If $F E I_c$ is not satisfiable, then no configuration exists that satisfies the feature model and soft constraints with I_c being true. I_c is therefore hidden by c .

IV. PATTERNS OF SOFT CONSTRAINT USE

In this section, we propose some typical patterns for the use of soft constraint annotations in feature models, with respect to the topological structure of the annotated feature model. We strived for identifying such patterns for multiple reasons:

- Identifying typical patterns of use improves understanding of the subject matter and provides insight into potential applications of soft constraints.
- If typical patterns of application are identified, with respect to the topological structure of the feature model, it becomes possible to automatically annotate feature models with such constraints for the purposes of generating test cases for experimenting and validating the automated analysis techniques we describe.

In this way, we have identified three patterns that describe specific cases of application of soft constraints. Naturally, this list cannot be considered exhaustive in any way, but it is sufficient for the purposes of providing a basic understanding of soft constraint use and allowing the automated creation of test cases. The description of these three patterns follows:

A. Soft Constraint Pattern: Reversed Constraint Suggestion

The pattern *Reversed Constraint Suggestion* (RCS) describes a situation where a feature model includes a hard constraint C , specifying a *requires* or *excludes* relation between two features (or their absence). The RCS of constraint C is a soft constraint that specifies that the reversed relation should also hold, that is, $RCS(A \Rightarrow B) = B$ suggests A .

Examples of RCS can be found in Fig. 5 and Fig. 6. The feature model in Fig. 6 is an adaptation of a simple automobile product line described in [9]. A hard constraint determines that the presence of the “Lateral Parking” feature requires selection of the “Lateral Range Finder” feature. The RCS of this hard constraint can be found in the feature model: using the lateral range finder feature suggests the use of lateral parking. In Fig. 5, the “Basic *absence-suggests* GPS” soft constraint is the RCS of the “GPS *excludes* Basic”.

The conceptual interpretation of the *RCS* pattern is based on the intuitive notion that, in some cases, if all the requirements for a certain feature (as specified by hard constraints) are met, then it may be sensible to opportunistically select it. For example, in Fig. 6 example, the suggestion of selecting “Lateral Parking” in the presence

of the “Lateral Range Finder” sensor can be interpreted as a suggestion to take maximum advantage of all the potential capabilities provided by the installed hardware in the vehicle. A similar perspective is that the “Lateral Range Finder” sensor can be considered of reduced utility if the “Lateral Parking” feature is not selected.

B. Soft Constraint Pattern: Group Selection Suggestion

The pattern *Group Selection Suggestion* (GSS) is related to the preferential configuration options of grouped features. Given a feature group G with subfeatures G_1, G_2, \dots , a GSS is a soft constraint that describes preferred group configurations (e.g.: $GSS(G) = \{G \Rightarrow G_x\}$). An example can be found in Fig. 7. The GSS represents the notion of preferred configuration options for feature groups.

C. Soft Constraint Pattern: Optional Selection Suggestion

The pattern *Optional Selection Suggestion* (OSS) encompasses a broad range of situations where domain-specific interdependencies affect the configuration of optional features. An OSS represents one such situation, by interlinking the configuration of two optional features via a soft constraint.

One such situation is represented in Fig. 5, where selection of the optional “Media” feature discourages selection of the optional “GPS” feature. This soft constraint could be understood to represent domain-specific constraints being embedded in the feature model: in this specific case, this soft constraint could be in place because of hardware performance limitations that could entail a non-negligible degradation of performance if both features are selected.

The OSS pattern represents domain-specific dependencies between optional features and as such has very generic scope. Specializations of this pattern may be devised if such domain-specific knowledge is considered. However, for the purposes of understanding typical structural patterns, it is sufficient.

V. AUTOMATED TEST CASE GENERATION

Although a large number of feature models can be obtained from online repositories [10], successfully applying these to the validation of the techniques proposed in this work entails the need to individually annotate the models with soft constraints. These models, however, are concerned with a large diversity of domains of application, making it extremely difficult to seek the help of independent domain experts for all relevant areas of expertise. Consequently, manual annotation may be feasible for only a relatively small number of models in specific areas of expertise. However, it is still a significantly time consuming task, where arbitrary decisions, that may put into question the credibility of the results, are unavoidable. Also, only a fraction of all available models may be considered, wasting a significant portion of potentially available resources and putting into question the representativity of any results that are obtained.

To address these difficulties, we have chosen to annotate all the feature models by automatically injecting soft constraints according to the usage patterns (RCS, GSS, and

OSS) described in Section VI. This approach has significant benefits:

- It allows any models in repository to be used for validation and test purposes.
- It speeds up test case generation significantly, and multiple configurations of soft constraint annotations for each model can be generated, allowing for a large test set to be created with the corresponding emergence of observable statistical properties and trends.

A. Test Case Generation

The following strategy was used to annotate a given base feature model with soft constraints according to the RCS, GSS, and OSS patterns.

Given the number of constraints N_c , number of groups N_G , the number of optional features N_O of the base feature model, and the configurable density parameters D_{RCS} , D_{GSS} , and D_{OSS} ,

1. Randomly select a constraint in the base feature model, generate the corresponding soft constraint according to RCS and insert it into the model. Do this $N_c * D_{RCS}$ times.
2. Randomly select one group in the base feature model and one subfeature belonging to that group. Generate the corresponding soft constraint according to GSS and insert it into the model. Do this $N_G * D_{GSS}$ times.
3. Randomly select two optional features in the base feature model, generate the corresponding soft constraint according to OSS and insert it into the model. Do this $N_O * D_{OSS}$ times.

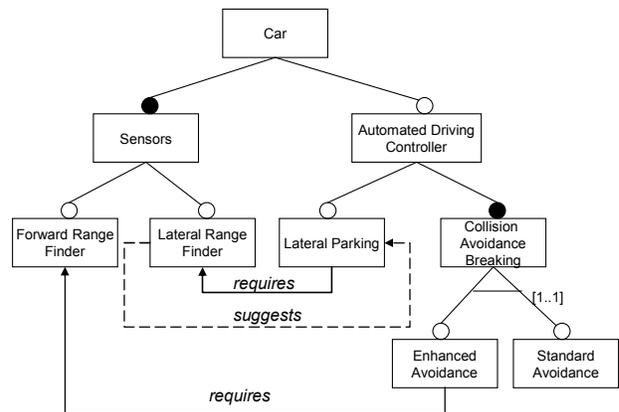


Figure 6. Example of a Reverse Constraint Suggestion, based on a feature model of a simple automobile product line.

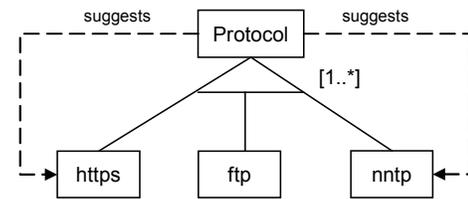


Figure 7. Example of Group Selection Suggestions

The density parameters control the number of soft constraints introduced. Nevertheless, fluctuations may occur because degenerate or nonsensical soft constraints (such as a feature suggesting one of its (grand-) parents) are ignored so they do not pollute or bias the results. Duplicate soft constraints may also be generated, so the actual number of soft constraints injected into the feature model may be less than $N_c * D_{RCS} + N_G * D_{GSS} + N_O * D_{OSS}$. This approach is effective and simpler than trying to always generate valid, distinct soft constraints in sufficient number (which may very well be impossible, depending on the chosen density parameters and structural properties of the base feature model).

VI. EXPERIMENTAL RESULTS

A. Tool description

We developed a Java based tool that processes and analyses feature models annotated with soft constraints. The SAT4J package was used for resolving satisfiability problems. The SXFM (Simple XML Feature Model) file format [10], used for storing feature model descriptions, was extended to include soft constraint information. The partial contents of a SXFM file containing soft constraint information can be observed in Fig. 8. Our tool is also capable of injecting soft constraints into feature models and conducting the analysis described in Section IV.

B. Test and Validation

For test and validation purposes, we have selected to use all the feature models with more than 40 features currently available in the SPLOT feature model repository [10]. These were provided by the site users and include models from both academic and industrial origin. This provided us with a large set of feature models with diverse characteristics and relevant dimension to validate our work.

Table III presents the relevant characteristics of the feature models. The descriptions were taken *verbatim* from their entries in the online feature repository.

```
<feature_tree>
:r Car
  :o Automated Driving Controller
  :m Collision Avoidance Braking
    :g [1,1]
      : Standard Avoidance
      : Enhanced Avoidance (enhanced_avoidance)
  :o Parallel Parking (parallel_parking)
  :m Sensors
    :o Lateral Range Finder (lateral_range_finder)
    :o Forward Range Finder (forward_range_finder)
</feature_tree>
<constraints>
c1: ~enhanced_avoidance or forward_range_finder
c2: ~parallel_parking or lateral_range_finder
</constraints>
<softconstraints>
s1: lateral_range_finder suggests parallel_parking
s2: forward_range_finder suggests enhanced_avoidance
</softconstraints>
</feature_model>
```

Figure 8. Extract of SXFM file extended with soft constraint information.

The purpose of the experiments is to observe to what extent the extent inclusion of soft constraints in feature models may lead to hidden features and inconsistencies as described in Section V, as well as assessing the effectiveness of the analysis algorithm. To this effect, soft constraints were injected in these models and the analysis algorithm was run to identify inconsistencies and hidden features. Although weak real world representativity is always a risk when using automated test case generation, this concern is mitigated by employing typical patterns of usage to guide soft constraint injection.

Different test sets were created by injecting soft constraints with increasing density parameters D_{RCS} , D_{GSS} , and D_{OSS} . All density parameters were set to the same value in each test set, and four different test sets were created, with density values of 0.125, 0.25, 0.5, and 0.75.

The results in Fig. 9 represent the aggregate results of running soft constraint injection and analysis for the models in Table III, while Table IV presents the results for each individual model. Because feature injection is a stochastic process, experiments were run 5 times for each feature model for each different setting of the density parameters, for a total of 20 runs per feature model.

The injection algorithm fails to inject any soft constraint into three feature models (*Thread*, *Database Tool*, and *DS Sample*) at the lowest density setting, because of their specific topological properties. For preserving homogeneity, results for these three models are not represented in Table IV, since only higher density results are available; comparison with other results would not be meaningful.

Results in Fig. 9 illustrate that, as can be expected, inconsistencies noticeably increase with higher densities of soft constraints. The number of inconsistencies seems to increase linearly with the number of soft constraints, while the number of orthogonal constraint sets increases more rapidly and appears to converge to a value in the vicinity of 80%. The number of unaffected feature models decreases correspondingly, until it drops below the number of inconsistencies at densities of approximately 65%. An important observation is that a significant number (approximately 20%) of inconsistent soft constraints is found even for low densities of soft constraints. This highlights the usefulness of automated analysis procedures for validating feature models annotated with soft constraints.

Results in Table IV show that adding soft constraints to two specific feature models (*PFTest1* and *DELL Laptop/Notebook Computers*) systematically resulted in the appearance of an inconsistent set of soft constraints set. Analyzing the characteristics of these two models, it is easily observed that one common distinguishing feature is the very high number of hard constraints in each (even after normalizing according to the number of features). It can be concluded that constraint density, and not feature model dimension or other factors, is the main contributing factor for the appearance of inconsistent soft constraint suggestions.

Hidden features were also identified. Table V presents the percentage of soft constraints hiding a feature as a percentage of the total number of soft constraints. For most feature models, the percentage of hidden features increases

with the density of soft constraints. The average results displayed in the last row of Table IV confirm this tendency. No hidden features were identified in *PFTest1* or *DELL Laptop/Notebook Computers*, for the simple reason that those feature models systematically generated inconsistencies precluding satisfaction of constraints. The presence of hidden implicants is found to be prevalent enough in most models so that automatic detection and report can be considered useful.

The analysis algorithm was found to be very efficient. Experiments were conducted in a netbook with 1 Gb of RAM memory, taking approximately 5-10s for loading, injecting soft constraints and analyzing once every one of the feature models considered in this test (between 0.15 and 0.3 s per feature model).

VII. RELATED WORK

In [6], probabilistic feature models are described that use soft constraints as descriptions of features that have high probabilities of being concurrently selected in the same configuration. Probabilistic feature models and corresponding samples spaces are suited to represent feature models obtained through feature mining processes, because complete feature joint probability distributions must be obtained. Incomplete specifications must be handled by complementary mechanisms such as Bayesian networks. In our work, the use of standard Boolean propositional logic capitalizes on established tool support and improves accessibility to the developer. This allows soft constraints to be more readily used to represent important domain knowledge in feature models.

“Encourages” and “discourages” constraints have been proposed for feature models in [11]. However, no precise semantics have been provided, precluding automated analysis and reasoning as described in our work.

In [12], fuzzy logic is applied to relate feature configurations to customer profiles. Although it is a significant departure from standard feature modeling approaches familiar to developers, Fuzzy logic is a powerful alternative tool for handling uncertainty.

While we focus this work on detection of inconsistencies and semantical analysis (e.g: detection of hidden features) of feature models annotated with Boolean soft constraints, in [7] improved configuration support is described.

Soft constraint frameworks have been studied in the context of constraint programming. These approaches focus on the search of a optimal variable assignment with respect to a set of quantified soft constraint expressions, as opposed to semantical and consistency analysis [13].

VIII. CONCLUSIONS

We have experimentally demonstrated the usefulness and viability of automated analysis of soft constraints in feature models. Typical patterns of soft constraint use were described. These were injected in publicly available feature models. In this process, inconsistencies and hidden features were introduced. These situations can correspond to potential semantic errors and should be reported back to the user for further inspection. Our tool was applied and was effective in

identifying these potential problems. This demonstrates that a framework for handling soft constraints in feature models using propositional logic can be a valuable tool for feature modeling. Future work will be conducting in identifying additional patterns of soft constraint use. The role of soft constraint usage in typical development tasks such as refactoring or domain modeling will also be investigated.

TABLE III. FEATURE MODELS

Description	Number of Features	Optional Features	Number of Groups	Hard Constraints
AndroidSPL	45	8	9	5
Arcade Game PL	61	4	9	34
bCMS system	66	6	8	2
Billing	88	45	2	59
Car Selection	72	10	19	21
Consolas de Videojuegos	41	11	2	5
Database Tool	40	7	7	0
DATABASE_TOOLS	70	20	7	2
DELL Laptop/Notebook Computers	46	1	8	110
Documentation_Generation	44	3	9	8
DS Sample	41	0	6	0
Electronic Drum	52	1	11	0
E-science application	61	7	16	2
HIS	67	10	6	4
Hotel Product Line	55	31	7	0
J2EE web architecture	77	26	11	0
Letovanje	43	3	13	2
Linea de Experimentos	52	11	4	4
Meshing Tool Generator	40	8	11	17
Model_Transformation	88	11	25	0
OW2-FraSCAti-1.4	63	39	2	46
PFTest1	56	5	8	90
Plone Meeting	57	13	9	0
Printers	172	1	28	0
Reuso – UFRJ – Eclipse1	72	40	7	1
Smart Home	56	36	4	0
Smart Home v2.2	60	30	6	2
SmartHome_vConejero	59	33	0	3
SPL SimulES, PnP	59	8	14	0
Thread	44	0	7	0
Video Player	53	17	9	2
Video Player	71	12	5	0
Web_Portal	43	17	6	6
Xtext	137	95	0	1

TABLE IV. INCONSISTENCY RESULTS PER FEATURE MODEL

Description	Unaffected	Orthogonal Inconsistent	Non-orthogonal Inconsistent
Model_Transformation	50%	50%	0%
OW2-FraSCAti-1.4	25%	75%	0%
Documentation_Generation	65%	35%	0%
SPL SimulES, PnP	45%	55%	0%
PFTest1	0%	0%	100%
DELL Laptop/Notebook Computers	0%	0%	100%
Linea de Experimentos	50%	50%	0%
Letovanje	50%	45%	5%
Xtext	0%	100%	0%
Smart Home v2.2	30%	60%	10%
SmartHome_vConejero	20%	80%	0%
bCMS system	60%	40%	0%
E-science application	50%	50%	0%
DATABASE_TOOLS	40%	60%	0%
Reuso - UFRJ - Eclipse1	55%	45%	0%
Hotel Product Line	20%	75%	5%
Electronic Drum	30%	70%	0%
Video Player	55%	45%	0%
Billing	40%	60%	0%
Smart Home	40%	60%	0%
Plone Meeting	25%	75%	0%
Meshing Tool Generator	15%	85%	0%
AndroidSPL	40%	60%	0%
Arcade Game PL Feature Model	40%	40%	20%
Web_Portal	35%	60%	5%
Consolas de Videojuegos	95%	5%	0%
Car Selection	40%	60%	0%
Printers	50%	35%	15%
HIS	30%	10%	60%
J2EE web architecture	30%	70%	0%

TABLE V. HIDDEN FEATURE RESULTS PER FEATURE MODEL

Description	Percentage of constraints hiding features			
	density 0,125	density 0,25	density 0,5	density 0,75
Model Transformation	0%	0%	0%	79%
OW2-FraSCAti-1.4	4%	56%	98%	91%
Documentation_Generation	0%	0%	11%	10%
SPL SimulES, PnP	0%	33%	67%	69%
PFTest1	0%	0%	0%	0%
DELL Laptop/Notebook Computers	0%	0%	0%	0%
Linea de Experimentos	0%	0%	0%	20%
Letovanje	0%	11%	29%	36%
Xtext	3%	6%	10%	21%
Smart Home v2.2	0%	0%	32%	33%
SmartHome_vConejero	0%	0%	30%	32%
bCMS system	0%	33%	13%	0%
E-science application	0%	0%	21%	63%
DATABASE_TOOLS	0%	0%	8%	49%
Thread	-	0%	22%	47%
Reuso - UFRJ - Eclipse1	0%	6%	30%	28%
Hotel Product Line	0%	0%	17%	11%
Database Tool	-	0%	6%	43%
Electronic Drum	0%	0%	67%	100%
Video Player	0%	0%	0%	6%
Billing	45%	41%	100%	100%
Smart Home	0%	0%	8%	9%
Plone Meeting	0%	7%	10%	33%
Meshing Tool Generator	0%	21%	0%	14%
AndroidSPL	0%	13%	58%	88%
Arcade Game PL Feature Model	0%	0%	0%	2%
Web_Portal	0%	11%	27%	32%
Consolas de Videojuegos	0%	22%	21%	25%
Car Selection	0%	0%	15%	0%
Printers	0%	0%	43%	41%
DS Sample	-	0%	0%	67%
HIS	0%	8%	27%	12%
J2EE web architecture	0%	8%	6%	20%
AVERAGE	2%	8%	23%	36%

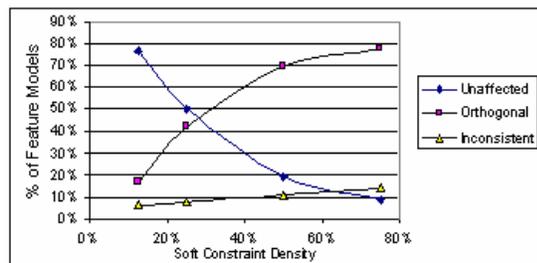


Figure 9. Aggregate results: unaffected, orthogonal and inconsistent feature models.

REFERENCES

[1] J. Barreiros and A. Moreira, "Soft Constraints in Feature Models," in *International Conference in Software Engineering Advances, ICSEA'11* Barcelona, 2011.

[2] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*: Addison-Wesley Professional, 2000.

- [3] P. Clements and L. Northrop, *Software Product Lines:Practices and Patterns*: Addison-Wesley, 2001.
- [4] D. Batory, "Feature-Oriented Programming and the AHEAD Tool Suite," in *26th International Conference on Software Engineering*: IEEE Computer Society, 2004.
- [5] K. Czarnecki and A. Wasowski, "Feature Diagrams and Logics: There and Back Again," in *11th International Software Product Line Conference (SPLC) Kyoto*, 2007, pp. 23-34.
- [6] K. Czarnecki, S. She, and A. Wasowski, "Sample Spaces and Feature Models: There and Back Again," in *Software Product Lines, 12th International Conference, SPLC Limerick*, Ireland, 2008, pp. 22-31.
- [7] J. Barreiros and A. Moreira, "Configuration Support for Feature Models with Soft Constraints " in *ACM Symposium on Applied Computing (in press)* Coimbra, 2013.
- [8] M. Mendonça, A. Wasowski, and K. Czarnecki, "SAT-based analysis of feature models is easy," in *Software Product Lines, 13th International Conference, SPLC 2009*, San Francisco, California, USA, 2009, pp. 231-240.
- [9] J. White, B. Dougherty, and D. C. Schmidt, "Automated reasoning for multi-step feature model configuration problems," in *Software Product Line Conference 2009* San Francisco, USA, 2009.
- [10] M. Mendonça, M. Branco, and D. Cowan, "S.P.L.O.T - Software Product Lines Online Tools," in *24th ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages and Applications, OOPSLA 2009* Orlando, USA, 2009.
- [11] H. Wada, J. Suzuki, and K. Oba, "A feature modeling support for non-functional constraints in service oriented architecture.," *IEEE Computer Society*, pp. 187-195, 2007.
- [12] S. Robak and A. Pieczynski, "Employment of fuzzy logic in feature diagrams to model variability in software families.," in *10th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2003)* Huntsville, AL, USA, 2003, pp. 305-311.
- [13] F. Rossi, P. V. Beek, and T. Walsh, "Handbook of Constraint Programming," in *Foundations of Artificial Intelligence*, J. Hendler, H. Kitano, and B. Nebel, Eds.: Elsevier, 2006.

Using Functional Complexity Measures in Software Development Effort Estimation

Luigi Lavazza

Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
luigi.lavazza@uninsubria.it

Gabriela Robiolo

Departamento de Informática
Universidad Austral
Buenos Aires, Argentina
grobiolo@austral.edu.ar

Abstract — Several definitions of measures that aim at representing the size of software requirements are currently available. These measures have gained a quite relevant role, since they are one of the few types of objective measures upon which effort estimation can be based. However, traditional Functional Size Measures do not take into account the amount and complexity of elaboration required, concentrating instead on the amount of data accessed or moved. This is a problem since the amount and complexity of the required data elaboration affect the implementation effort, but are not adequately represented by the current size measures, including the standardized ones. Recently, a few approaches to measuring aspects of user requirements that are supposed to be related with functional complexity and/or data elaboration have been proposed by researchers. In this paper, we take into consideration some of these proposed measures and compare them with respect to their ability to predict the development effort, especially when used in combination with measures of functional size. A few methods for estimating software development effort –both based on model building and on analogy– are experimented with, using different types of functional size and elaboration complexity measures. All the most significant models obtained were based on a notion of computation density that is based on the number of computation flows in functional processes. When using estimation by analogy, considering functional complexity in the selection of analogue projects improved accuracy in all the evaluated cases. In conclusion, it appears that functional complexity is a factor that affects development effort; accordingly, whatever method is used for effort estimation, it is advisable to take functional complexity into due consideration.

Keywords – *Functional size measurement; Function Points; COSMIC function points; effort estimation; functional complexity measurement.*

I. INTRODUCTION

Several definitions of measures intended to represent the functional size of software are being used. The popularity of these measures is due to the fact that functional size measures are typically used to drive the estimation of the development effort. To this end, effort models require several inputs in addition to the functional size, including the complexity of the software to be developed [11][47]. In fact, problem complexity is recognized as one of the elements that contribute to the comprehensive notion of software size [17].

The need to account for software complexity when estimating the development effort does not depend on the functional size measurement (FSM) method used.

Before proceeding, it is useful to spend some words on the fact that throughout the paper we treat the terms “complexity” and “amount of data elaboration” as synonyms. This is due to the facts that complexity is an inherently elusive concept, and at the functional requirements level it is not clear what should be the difference between the amount and the complexity of data elaboration: for instance, in many cases, complexity is considered proportional to the number of alternatives in a process execution, but this number is also clearly related also to the size of the process.

When dealing with effort estimation, the most popular methods require an evaluation of the complexity of the application. Currently such evaluation is of a purely qualitative nature. For instance, COCOMO II [11] provides a table that allows the user to evaluate complexity on an ordinal scale (from “very low” to “extra high”) according to five aspects (control operations, computational operations, device-dependent operations, data management operations, user interface management operations) that have to be evaluated in a qualitative and subjective way: e.g., the characterization of computational operations corresponding to the “Nominal” complexity is “Use of standard math and statistical routines. Basic matrix/vector operations” [15].

It is quite clear that it would be greatly beneficial to replace such subjective and approximate assessment of complexity with a real measure, based on objective and quantitative evaluations, since this would enable the construction of more reliable and accurate models of effort.

Previous work showed that effort models that take into consideration complexity measures are more precise than those based on the functional size only. In particular, the authors of this paper showed that development effort correlates well with COSMIC function points (CFP) [16] and Path [43], and that the inclusion of a Path-based complexity measure improves the models based on size, whatever size measure is used (IFPUG Function Points [24], CFP [23], or even Use Case Points [28]) [34].

In [1], the work reported in [34] was extended, by taking into consideration some measures that represent potential complexity dimensions, by building effort estimation models that exploit these measures, and by discussing the precision of fit of these models.

In this paper, we further enhance the work reported in [1] by using an extended dataset, and by refining it (the largest project was removed, being an evident outlier). More important, here we test the importance of functional complexity measures in effort estimation, by experimenting with a wider range of estimation methods. In particular, we use not only model-based estimation, but also Estimation by Analogy (EbA), as this is a very popular technique: model-based estimation and EbA are definitely the most relevant techniques for cost estimation [25].

The results of the measurements and analyses reported in the paper contribute to enhancing the knowledge of how to measure functional complexity at the requirements level, and what is the contribution of such measure to effort estimation.

The paper is organized as follows: Section II accounts for related work; Section III is dedicated to illustrating the measures of functional size and functional complexity used in this study; Section IV describes the dataset and the types of analysis performed; Section V and VI illustrate the results of the analyses via regression and analogy, respectively; in Section VII the outcomes of the research are discussed; in Section VIII the threats to the validity of the study are discussed. Finally, Section IX draws some conclusions and outlines future work.

II. RELATED WORK

A few attempts to account for data elaboration in FSM have been done. Feature points by Capers Jones [26] aim at capturing the algorithmic complexity of the elaboration. However, according to Capers Jones, “the feature point metric was created to deal with the psychological problem that members of the real-time and systems software world viewed function point metrics as being suitable only for management information systems” [27]. Therefore, feature points simply moved part of the ‘size’ from data to algorithms, leaving the measure substantially unaltered with respect to FPA. In fact, currently Capers Jones recommends “the use of the standard IFPUG methodology combined with a rating of ‘Project Complexity’ to properly scale effort”.

3D Function Points [50] 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.

Gencil and Demirors [19] point out that we still need a new Base Functional Component (BFC) Types for the Boolean operations of Functional User Requirements, which are often not considered to be algorithmic operations, but which are related to complexity. This point of view highlights the necessity of considering the complexity of elaboration required in FSM, and they suggested introducing as a new BFC type that differs from authors’ proposal.

Bernárdez et al. [10] measured the cyclomatic complexity of a use case in order to validate the use case definition, while Levesque et al. [35] measured the conditions of inputs in a sequential diagram in order to add the concept of complexity to the COSMIC method.

Yavari et al. [51] evaluated the weak points of Use Case complexity measures, in particular, those of transaction identification, and introduced other measures to determine Use Case complexity. They focused on Use Case specification and flow of events. Also, the authors considered main and alternative scenarios. However, this is only an early definition of the new measures, as they did not use them in a case study.

Aggarwal et al. [4] defined an estimation model that can be used to estimate the effort required for designing and developing hypermedia content management systems (CMS). The model is designed to help project manager to estimate effort at the very early stage of requirement analysis. Questionnaires are used to estimate the complexity of the project. The final effort is estimated using the project size and various adjustment factors. The size of the project is evaluated by using a modified object point analysis approach. The proposed model shows a great improvement as compared to the earlier models used in effort estimation of CMS projects.

Visaggio [48] proposes a metric for expressing the entropy of a software system and for assessing the quality of its organization from the perspective of impact analysis. The metric is called “structural information” and is based on a model dependency descriptor. The metric is characterized by its independence from the techniques used to build the system and the architectural styles used to represent the system at the various levels of abstraction. The metric is sensitive to and reflects both internal and external complexity, but is independent of and ignores intrinsic complexity, which is our interest focus.

Briand and Wust [14] used structural design properties of an object-oriented development project, such as coupling, cohesion, and complexity (of late design) as additional cost factors. They empirically conclude that the measures of such properties did not play a significant role in improving system effort predictions.

Mendes et al. [38] compared length, functionality and complexity metrics as effort predictors by generating corresponding prediction models and comparing their accuracy using boxplots of the residuals for web applications. Their results suggest that in general the various considered measures provide similar prediction accuracy.

Baresi and Morasca [8] analyzed the impact of attributes like the size and complexity of W2000 (a special-purpose design notation for the design of Web applications [7]) design artifacts on the total effort needed to design web applications. They identified for Information, Navigation, and Presentation models a set of size and complexity metrics. The complexity metrics are based on associations and links identified in the models. The three studies performed correlated different size measures with the actual effort: no general hypotheses could be supported by the analyses that were conducted, probably because the designer’s background impacted the perception of complexity.

Lind and Heldal [36] conducted four experiments in the automotive industry, which showed a strong correlation between COSMIC functional size measures and

implemented code size in Bytes of real-time applications. They reported that it was possible to obtain accurate Code Size estimates even for software components containing complex calculations –which are not captured by COSMIC– as the factors affecting the relationship are functionality type, quality constraints, development methods and tools, and information regarding hardware interfaces missing in the requirement specification.

Bashir and Thomson [9] used traditional regression analysis to derive two types of parametric models: a single variable model based on product complexity and a multivariable model based on product complexity and requirements severity. Generally, the models performed well according to a number of accuracy tests. In particular, product complexity explained more than 80% of variation in estimating effort. They concluded that product complexity as an indicator for project size is the dominant parameter in estimating design effort. Our results agree with those by Bashir and Thomson, as the results they obtained using functional complexity measures ($0.64 < R^2 < 0.81$) are quite similar to ours.

Quite interestingly, in the parametric models that are most used in practice –like COCOMO II [11] or SEER/SEM [18]– the functional complexity is taken into account as part of the product characteristics in formulas of the type $\text{Effort} = f(\text{Size}, \langle \text{product characteristics} \rangle, \langle \text{process characteristics} \rangle)$.

Hastings and Sajeev [21] proposed a Vector Size Measure (VSM) that incorporates both functionality and problem complexity in a balanced and orthogonal manner. VSM is used as the input to a Vector Prediction Model (VPM), which can be used to estimate development effort early in the software life cycle. The results indicate that the proposed technique allows for estimating the development effort early in the software life cycle with errors not greater than 20% across a range of application types.

AlSharif et al. [5] introduced a measure for assessing the overall complexity of software architecture. To accomplish this, they chose to use the Full Function Points (FFP) methodology –a former version of COSMIC Function Points– as a building block to measure complexity. The new measure was inspired by the fact that, in general, the components of an architecture comprise collections of services (functionality) that each component provides for other components. The allocation of these functionalities affects the required interface (external dependency) and the internal work performed by each component. Therefore, measuring the functionality of the components can serve as an indicator of the internal and external complexity of the components and, consequently, the complexity of the architecture. Also, Sengupta et al. [44] proposed the Component Architecture Complexity Measurement Metrics (CACMM), based on Component Architecture Graph (CAG), a graphical model used for representing a UML component diagram. An analysis of the graph was performed to measure complexity at different levels – the individual component level, the component-to-component level and the overall architecture. However, neither in [5]

nor in [44] the relationship between architecture complexity and effort was analyzed.

Misra [40] proposed a modified cognitive complexity measure (MCCM), which is a modification of the Cognitive Information Complexity Measure (CICM). In the cognitive functional size measure, the functional size depends upon the internal architecture of the software and its inputs and outputs. For the new measure, the occurrence of operators and operands is taken into account, instead of the number of inputs and outputs. The author compared the values obtained by calculating the complexity of eight C programs; however, a relation with Effort was not reported.

Wijayasiriwardhane and Lai [49] described a Function Point-like measure named Component Point (CP), which was used to measure the system-level size of a Component-Based Software System (CBSS), specified in the Unified Modeling Language. In the CP counting process, the complexity of the component was assessed, which depended not only on the number, but also on the complexity of its interfaces and interactions. The complexity level of each interface was specified using the Number of Operations (NO) and the Number of Parameters (NP), which were derived from the operation signatures for each interface. They provided an empirical analysis of seven projects in order to verify the validity and usefulness of the CP measure with regard to its correlation to the effort of component-based development. They reported that the R^2 obtained was greater than 0.9.

Our results are in accordance with the consideration expressed by Morasca on the definition of measures [42], as it appears that the notion of complexity may be represented by taking into account several basic indicators (size, control flow, data, etc.) that can be used individually (i.e., without the need to build a derived measure defined as a weighted sum) in estimation models.

Mittas and Angelis [41] introduced the use of a semi-parametric model that managed to incorporate some parametric information into a non-parametric model, combining in this way regression and analogy. They demonstrated the procedure used to build such a model from two well-known datasets. The MMRE reported for EbA were 35.57% and 33.45% and the improvement using the combination model was about 50%. The results using EbA fell within the range of our results, but the improvement obtained was higher. However, the method proposed by Mittas and Angelis has some limits in practical applicability, because the models are more difficult to build, as more variables and several estimation techniques have to be used.

Shepperd and Schofield [45] described an approach to estimation based upon the use of analogies. The underlying principle was to characterize projects in terms of features (for example, the number of interfaces, the development method or the size of the functional requirements document). Similarity was defined as the Euclidean distance in an n-dimensional space, where n is the number of project features. Each dimension is standardized, so all dimensions have equal weight. The known effort values of the closest neighbors to the new project are then used as the basis for

prediction. The method was validated on nine different industrial datasets (a total of 275 projects) and in all cases analogy outperformed algorithmic models based upon stepwise regression. Although we had a different research objective, it was useful to see that the results they obtained were in a range of values similar to ours: the MMRE of analogy based method of homogeneous data set were in the 26%-60% range.

Finally, Gupta et al. [20] studied analogy based estimation methods and compared estimation results reported by nine authors using Fuzzy logic, Grey System Theory, Machine Learning techniques such as Genetic Algorithms, Support vector Machines. The reported results are characterized by quite large variations: MMRE varies from a minimum 12% to a maximum 111%, while Pred(25) is in the 15%–83.75% range. Our results fall in these ranges of values.

III. MEASURES INVOLVED IN THE STUDY

In this study, we used the size measures and functional complexity measures described in the following sections.

A. Function Points

The Function Point method was originally introduced by Albrecht to measure the size of a data-processing system from the end-user's point of view, with the goal of estimating the development effort [2][3]. IFPUG FPA is now an ISO standard [24] in its "unadjusted" version. So, throughout the paper, unless otherwise explicitly stated, we refer exclusively to Unadjusted Function Points, which are generally referred to as "UFP."

The basic idea of FPA is that the "amount of functionality" released to the user can be evaluated by taking into account the data used by the application to provide the required functions, and the transactions (i.e., operations that involve data crossing the boundaries of the application) through which the functionality is delivered to the user. Both data and transactions are evaluated at the conceptual level, i.e., they represent data and operations that are relevant to the user. Therefore, Function Points (FP) are counted on the basis of the user requirements specification. The boundary indicates the border between the application being measured and the external applications and user domain.

The core of the counting procedure consists in identifying and weighting so-called data function types and transactional function types. Data functions represent data that are relevant to the user and are required to perform some function. Data functions (DF) are classified into internal logical files (ILF), and external interface files (EIF). An ILF is a user identifiable group of logically related information maintained (i.e., managed –in FPA terminology, "maintaining" data means creating, modifying, deleting data–) within the boundary of the application. An EIF is similar to an ILF, but is maintained within the boundary of another application, i.e., it is outside the application being measured, for which an EIF is a read-only file.

Transactional functions represent operations that are relevant to the user and cause input and/or output data to cross the application boundary. Transactional functions

represent elementary processes. An elementary process is the smallest unit of activity that is meaningful to the user(s). An elementary process must be self-contained and leave the application being counted in a consistent state. Transactional functions are classified into external inputs (EI), external outputs (EO), and external inquiries (EQ) according to the main intent of the process: updating ILF for EI, computing and outputting results for EO, retrieving and outputting data for EQ.

Every function, either data or transaction, contributes a number of FP that depends on its weight. The weight of ILF and EIF is evaluated based on Data Element Types (DET) and Record Element Types (RET). A DET is a unique, non-repeated field recognizable by the user. A RET is a subgroup of the information units contained in a file. To give a rough idea of what RET and DET are, if the specifications are written in an object-oriented language (like UML), the concept of RET maps (with some exceptions) onto the concept of class, while DET are the class attributes.

For transactions, the weight is based on the number of DET and File Type Referenced (FTR). An FTR can be an ILF referenced or maintained by the transaction or an EIF read by the transaction. The DET considered are those that cross the application boundary when the transaction is performed.

Each function is weighted according to given tables (weight tables can be found here: <http://www.eng-it.it/qg-ifpug-fpa-v42en.pdf>).

Finally, the number of so-called Unadjusted Function Points (UFP) is obtained by summing the contribution of the function types $UFP = EI + EO + EQ + ILF + EIF$.

According to the definition of UFP, it is clear that the amount and complexity of the elaboration required in the transaction functions is not taken into consideration. For instance two External Output transactions that involve 2 FTR and 10 DET both have the same weight, even if one just performs sums and the other performs very complex operations according to a very sophisticated algorithm.

B. COSMIC Function Points

COSMIC (Common Software Measurement International Consortium) [16][23] function points are growingly used for measuring the functional size of applications, i.e., to measure the size of functional user requirements.

COSMIC measurement is applied to the "Functional User Requirements" of a software application (actually, there is no difference between the user requirements used to count FP and CFP). The result is a number representing the functional size of the application in COSMIC Function Points.

In the COSMIC model of software (illustrated in Fig. 1), the Functional User Requirements can be mapped into unique functional processes, initiated by functional users. 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. In practice, the COSMIC data groups correspond to FPA logical data files (or to RET), but do not contribute directly to the size in CFP: they are

relevant only because they are the objects of data movements. There are four types of data movements:

- An Entry moves a data group into the software from a functional user.
- An Exit moves a data group out of the software to a functional user.
- A Read moves a data group from persistent storage to the software.
- A 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.

The size in CFP is given by equation $CFP = \text{Entries} + \text{Exits} + \text{Reads} + \text{Writes}$, where each term in the formula denotes the number of corresponding data movements. So, there is no concept of “weighting” of a data movement in COSMIC, or, equivalently, all data movement have the same unit weight.

COSMIC function points do not represent the amount and complexity of data elaboration required. COSMIC function points concentrate on the measure of data movements, neglecting data elaboration. More precisely, the model of software used by the COSMIC method –illustrated in Fig. 1– includes data elaboration, but no indication on how to measure it is provided. The COSMIC measurement manual [16] simply assumes that every data movement accounts for some amount of data elaboration, and that such amount is proportional to the number of data movements, so that by measuring data movements, one measures also data manipulation.

As size units, we adopted both CFP and the number of functional processes. In fact, the number of functional processes is suggested as a reasonable approximation of the size in CFP in [16]. Moreover, being a sort of “by product” of CFP measurement, computing the number of Functional Processes does not actually require additional effort.

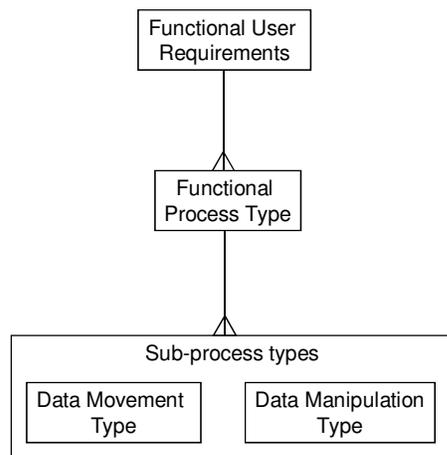


Figure 1. The COSMIC generic software model.

C. Use Case-based Measures

In 1993, Karner introduced Use Case Points (UCP), a redefinition of the Function Point method in the context of the use case requirements specification [28][29].

Use cases yield an observable result that is meaningful for the actors [12]. Karner based the conception of use case functional size on two elements: the use cases themselves and the actors, which are the external entities that interact with the system.

Every element, i.e., every use case and actor, contributes to the software product size with a number of UCP that depends on the “complexity” of each use case and actor. The complexity of use cases is defined in terms of the number of transactions and analysis objects (which are conceptually similar to ILF). The complexity of actors is associated to the characteristics of the interface and protocol used in the interaction with the system. Each element is weighted on the basis of its complexity according to the values specified in [28][29].

Since the concept of transaction is not clearly defined in Karner’s method –as already pointed out by [51]– we decided to interpret it as the stimulus triggered by an actor: each stimulus that an actor triggers defines a transaction, as described in [32]. Also, the analysis objects were not taken into account to measure the complexity of the use cases; only the number of transactions was used. This was done in order to make our results comparable to those obtained by [6].

We consider UCP in their “unadjusted” version. So, throughout the paper, unless otherwise explicitly stated, we will exclusively refer to Unadjusted Use Case Points, which are generally referred to as “UUCP.”

Finally, it is important to note that the number of UUCP, i.e., the “amount of functionality” of a use case, is obtained by adding up the contribution of the elements: $UUCP = \text{Use Case} + \text{Actor}$.

D. Functional Complexity Measures

Several different possible measures of functional complexity were proposed. For instance, in [46] the number of inputs and outputs, the number of decision nodes, the sum of predicates of all decision nodes, the depth of decision tree and the length of paths are considered as possible indicators of complexity.

In [47], Tran Cao et al. propose the usage of the number of data groups (NOD), the number of conditions (NOC) and entropy of system (EOS). They also study how these measures (also in combination with COSMIC FP) are correlated with the development effort.

Another measure of complexity, the Paths, was defined on the basis of the information typically available from use case descriptions [43]. The measure of the complexity of use cases is based on the application of the principles of McCabe’s complexity measure [37] to the descriptions of use cases in terms of scenarios. In fact, use cases are usually described giving a main scenario, which accounts for the ‘usual’ behaviour of the user and system, and a set of alternative scenarios, which account for all the possible deviations from the normal behaviour that have to be

supported by the system. Robiolo and Orosco [43] apply to the use case textual descriptions the same measure applied by McCabe to code. Every different path in a given use case scenario contributes to the measure of the use case's complexity. The definition of Paths conforms to several concepts enounced by Briand et al. [13]: Paths represent "an intrinsic attribute of an object and not its perceived psychological complexity as perceived by an external observer", and they represent complexity as "a system property that depends on the relationship between elements and is not an isolated element's property". A detailed description of the Paths measure and its applicability to use cases described in UML can be found in [33].

In the research work reported here, we used measures that are conceptually very close to those proposed in previous studies [46][47]. However, we did not stick exactly to the previous proposals, essentially for practical reasons. We used Paths instead of NOC because both measures capture essentially the same meaning, and the measures of Paths were already available. Similarly, we used the number of COSMIC data groups and the FPA number of logic data files instead of NOD because –having measured the size of the applications in FP and CFP, the documentation on data groups and logic files was already available, thus the measurement could be performed very easily.

TABLE I. THE DATASET

Project ID	Type	Actual Effort	Path	Use Cases	UUCP	UFP	FPA transactions	CFP	Functional Processes	Data Groups	Pers. DG
P1	Academic	410	71	39	201	185	39	143	39	21	7
P2	Academic	474	73	28	149	269	58	118	28	15	9
P3	Academic	382	60	7	84	171	19	109	24	15	12
P4	Academic	285	49	6	72	113	15	74	25	14	8
P5	Academic	328	34	12	72	110	14	48	12	17	7
P6	Academic	198	35	8	62	86	9	67	10	15	7
P7	Academic	442	50	6	71	75	10	81	16	12	6
P8	Industrial	723	97	27	175	214	33	115	27	19	10
P9	Industrial	392	83	15	111	340	47	105	24	35	24
P10	Industrial	272	42	19	119	179	27	73	21	9	9
P11	Industrial	131	18	13	68	115	17	51	13	5	5
P12	Industrial	348	32	12	71	107	16	46	12	13	7
P13	Academic	243	68	12	99	111	12	96	26	18	9
P14	Academic	300	33	4	57	40	4	54	12	12	4
P15	Academic	147	20	10	53	59	10	53	14	15	4
P16	Academic	169	17	5	28	61	5	30	5	10	6
P17	Academic	121	21	13	52	72	13	47	13	15	5
P18	Academic	342	24	9	48	11	27	40	9	12	2
P19	Academic	268	16	9	49	12	27	30	9	10	3

IV. THE EXPERIMENTAL EVALUATION

A. The Dataset

In order to evaluate the measures mentioned above with respect to their usability as effort predictors, we collected all such measures for a set of projects. We could not use data from the best known repositories –such as the PROMISE or ISBSG datasets– because they do not report the size of each project according to different FSM methods; moreover, the Paths measure is quite recent, and no historical data exist for it.

We measured 19 small business projects, which were developed in three different contexts: an advanced undergraduate academic environment at Austral University,

the System and Technology (S&T) Department at Austral University and a CMM level 4 Company. The involved human resources shared a similar profile: advanced undergraduate students, who had been similarly trained, worked both at the S&T Department and at the CMM level 4 Company. All the selected projects met the following requisites:

- Use cases describing requirements were available.
- All projects were new developments.
- The use cases had been completely implemented, and the actual development effort in PersonHours was known.

The dataset is reported in Table I. Note that we distinguished the number of persistent data groups (column *Pers. DG*) from the total number of data groups, which includes also transient data groups. Our hypothesis is that

persistent data groups are more representative of the amount of data being handled by the application.

B. The Estimation Methods Used

We first checked if statistically significant models could be built using ordinary least squares (OLS) linear regression.

We used also linear regression after log-log transformation, as is usually done in studies concerning effort [11][47].

In model building, a 0.05 statistical significance threshold was used throughout the paper, as is customary in Empirical Software Engineering studies. All the results reported in the paper are characterized by p-value < 0.05. All the validity requirements for the proposed models (e.g., the normal distribution of residuals of OLS regressions) were duly checked.

Moreover, in order to avoid overfitting, we retained only models based on datasets containing –after the elimination of outliers– at least seven data points for each independent variables. In this way we get datasets containing enough data points to support statistically significant analyses.

Then, we applied Estimation by Analogy (EbA). That is, we estimated the development effort required by each project on the basis of the actual development effort required by similar projects (i.e., projects having similar size and/or complexity characteristics).

C. The Measures

The measures actually employed in the analysis are described in Table II. They are a superset of those used in [1].

Among the measures listed in Table II, we have not only size and complexity measures, but also several density measures. These density measures introduce the concept of functional complexity per size unit: for instance, we consider the Path/UFP ratio, which indicates how many Paths per UFP are required in the software being measured. The reason for using functional complexity per size unit is that functional complexity measures are often correlated to size itself, thus the density appears more relevant to indicate how complex and difficult the development is.

The complexity of a system is a property that depends on the relationships among system's elements [13]. So, the measures discussed above represent the density of relationships among elements per unit size.

V. USING FUNCTIONAL COMPLEXITY MEASURES IN EFFORT ESTIMATION MODELS

In this section, we report about the construction of development effort models. We systematically tried to build models that have the development effort as dependent variable and one or two independent variables belonging to the set described in Table II. In particular, when building models having two independent variables, all the possible pairs of measures from Table II were tried out. Models with more than two independent variables were not sought, because the available dataset does not contain enough data points to support such analysis while avoiding data overfitting.

TABLE II. MEASURES USED

Name	Description
Path	Path
CFP	COSMIC Function Points
FPr	Functional Processes
DG	Data Groups
PDG	Persistent Data Groups
UC	Number of Use Cases
UUCP	Unadjusted Use Case Points
UFP	Unadjusted Function Points
FPAtrans	Number of unweighted FPA transactions
NumFiles	Number of unweighted FPA logic data files
Path/FPr	Path per Functional Process
Path/CFP	Path per CFP
DG/FPr	Data Groups per Functional Process
DG/CFP	Data Groups per CFP
PDG/FPr	Persistent Data Groups per Functional Process
PDG/CFP	Persistent Data Groups per CFP
Path/UFP	Paths per unadjusted Function Point
Path/FPAtrans	Paths per unweighted FPA transactions
NumFiles/UFP	Unweighted FPA logic data files per unadjusted Function Point
NumFiles/FPAtrans	Unweighted FPA logic data files per unweighted FPA transactions
Path/UC	Paths per use case
Path/UUCP	Paths per use case point

A. Analysis of the dataset using linear regression

Linear regression did not provide any statistically significant model when FPA or UC measures were used. On the contrary, when using COSMIC-based measures, a single statistically significant model was found, having equation

$$\text{Effort} = -29.9 + 139.1 \times \text{Path/FPr}$$

The model –obtained after eliminating 4 outliers (P1, P6, P8, P17)– has adjusted $R^2=0.64$. The regression line is illustrated in Fig. 2. This model is quite interesting, as a functional size density measure is the independent variable: the model seems to suggest that the actual complexity of software, rather than its size, determines development effort.

The accuracy of the model is characterized by MMRE = 34%, while Pred(25) –i.e., the percentage of project whose absolute relative estimation error is in the $\pm 25\%$ range– is 63%, and Error range = -46%–162%. The distribution of relative residuals is illustrated in Fig. 3.

Throughout the paper, MMRE and Pred(25) are used as indicators of the accuracy of models, because they are often quoted as the *de facto* current accuracy indicators used in

Empirical Software Engineering, even though criticisms have been cast on the usefulness and meaning of MMRE and Pred(25) [31]. Boxplots representing the distributions of relative residuals are always given, to provide meaningful and unbiased information on estimation accuracy.

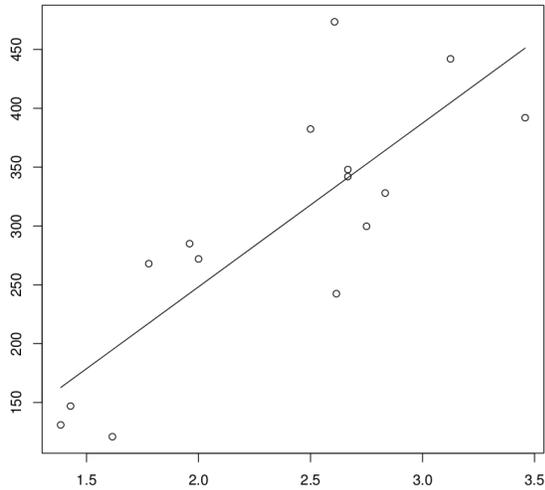


Figure 2. Effort vs. Path/FPr: OLS regression line.

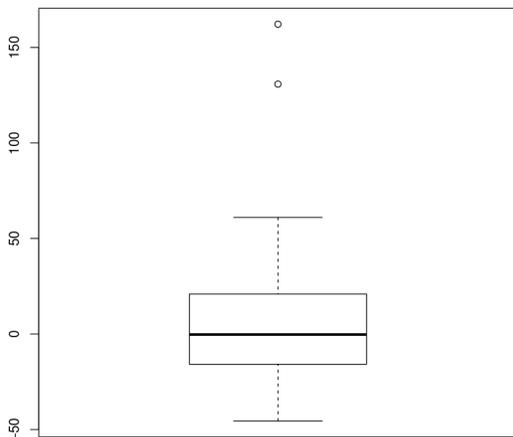


Figure 3. Effort vs. Path/FPr OLS model: relative residuals.

B. Analysis of the dataset using log-log transformations

We used linear regression after log-log transformation, as is usually done in studies concerning effort [11][47].

The models found by checking the correlation between effort and FPA measures are summarized in Table III, while their accuracy is illustrated in Table IV. Both the models found involve a size measure (the number of FPA transactions) and a complexity density measure (Paths/UFP or Path/FPA transactions).

TABLE III. OLS EFFORT MODELS INVOLVING FPA-BASED MEASURES

Model	Adj. R ²	Outl.
$\text{Effort} = 120.6 \times \text{FPATrans}^{0.654} \times (\text{Path/UFP})^{0.958}$	0.642	3 (P21, P22, P17)
$\text{Effort} = 12.2 \times \text{FPATrans}^{0.816} \times (\text{Path/FPATrans})^{0.976}$	0.812	3 (P22, P21, P14)

TABLE IV. ACCURACY OF OLS FPA-BASED EFFORT MODELS

Model	MMRE	Pred(25)	Error range
$\text{Effort} = 120.6 \times \text{FPATrans}^{0.654} \times (\text{Path/UFP})^{0.958}$	71%	53%	-40%–543%
$\text{Effort} = 12.2 \times \text{FPATrans}^{0.816} \times (\text{Path/FPATrans})^{0.976}$	25%	63%	-60%–108%

The models found by checking the correlation between effort and COSMIC measures are summarized in Table V. It is interesting to note that both the models found involve the usage of a size measure (the Function Points or the number of functional processes) and a complexity density measure (Paths/CFP). The accuracy of the models found is illustrated in Table VI.

TABLE V. OLS EFFORT MODELS INVOLVING COSMIC-BASED MEASURES

Model	Adj. R ²	Outliers
$\text{Effort} = 112.2 \times \text{CFP}^{0.391} \times (\text{Path/CFP})^{1.298}$	0.658	0
$\text{Effort} = 231.8 \times \text{FPr}^{0.377} \times (\text{Path/CFP})^{1.468}$	0.744	1 (P14)

TABLE VI. ACCURACY OF OLS COSMIC-BASED EFFORT MODELS

Model	MMRE	Pred(25)	Error range
$\text{Effort} = 112.2 \times \text{CFP}^{0.391} \times (\text{Path/CFP})^{1.298}$	22%	68%	-30%–77%
$\text{Effort} = 231.8 \times \text{FPr}^{0.377} \times (\text{Path/CFP})^{1.468}$	23%	68%	-27%–97%

Finally, we checked the correlation between effort and Use Case. The models found are summarized in Table VII. It is noticeable that both the models found involve the usage of a size measure (either the number of use cases or the use case points) and a complexity density measure.

TABLE VII. OLS EFFORT MODELS INVOLVING USE CASE-BASED MEASURES

Model	Adj. R ²	Outliers
$\text{Effort} = 21.9 \times \text{UC}^{0.68} \times (\text{Path/UC})^{0.728}$	0.609	1 (P22)
$\text{Effort} = 24.9 \times \text{UUCP}^{0.679} \times (\text{Path/UUCP})^{0.775}$	0.608	1 (P22)

The accuracy of the models found is illustrated in Table VIII. Again, we got significant models only based on variables involving Paths.

It is quite interesting to observe that all the models found have two independent variables, and that one is a measure of size, while the other is a measure of complexity density.

TABLE VIII. ACCURACY OF OLS USE CASE-BASED EFFORT MODELS

Model	MMRE	Pred(25)	Error range
$\text{Effort} = 21.9 \times \text{UC}^{0.68} \times (\text{Path/UC})^{0.728}$	24%	63%	-45%–73%
$\text{Effort} = 24.9 \times \text{UUCP}^{0.679} \times (\text{Path/UUCP})^{0.775}$	24%	63%	-45%–74%

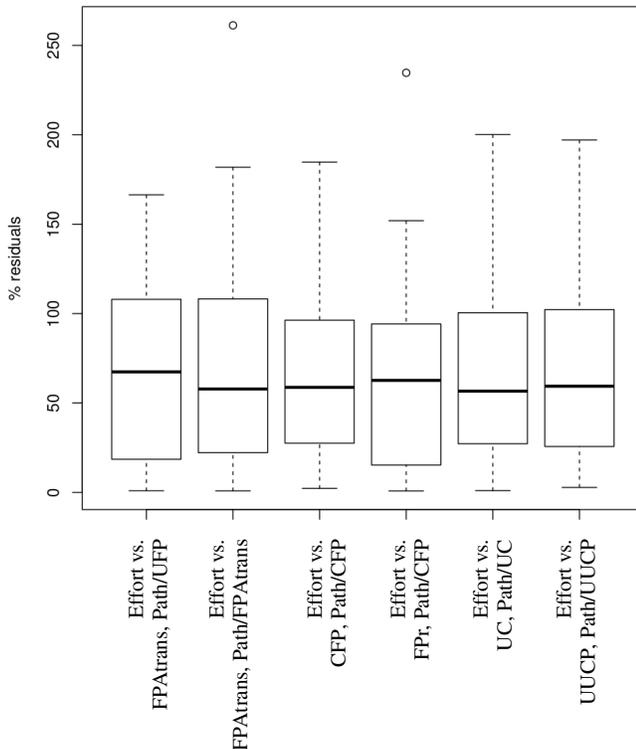


Figure 4. Model comparison: relative residuals.

It is also interesting to see that these models appear reasonably good both in terms of their ability to explain the variation of effort depending on the variation of the size and complexity measures (as indicated by the values of the adjusted R^2) and in terms of precision of the fit (as indicated by MMRE, Pred(25) and the relative error range).

The data illustrated in the tables above do not indicate if a model is definitely better than the others with respect to accuracy. A possible way for identifying the best model is by comparing the relative residuals (since we are considering the ability to predict effort, we have to look at *relative* residuals, since an error of, say, two PersonMonths can be irrelevant or very important, depending on the total effort).

The boxplots representing relative residuals of the models obtained via OLS regression after log-log transformation are reported in Fig. 4 (where two extreme outliers of the first model were omitted to preserve the readability of the figure). The boxplots indicate quite clearly that the values and distributions of residuals are very similar for all the models.

VI. USING FUNCTIONAL COMPLEXITY MEASURES IN ESTIMATION BY ANALOGY

In this section, we test the effects of considering complexity density when selecting “analogous” projects upon which effort estimation is based. To this end, we compute effort estimates in two ways: identifying analogous projects only on the basis of size, and considering both size and complexity density. The hypothesis that we want to test

is that considering both size and complexity density leads to better estimates.

There are several criteria that can be used to identify analogous projects. Some of these involve identifying the k closest projects, where closeness is generally evaluated as the distance between projects. Accordingly, we chose to use a criterion that ensures –as far as possible– that the analogous projects are all within a given distance from the project to be estimated.

For each project, we select among the remaining projects those having both size and complexity density in a $\pm 20\%$ range. The mean of these projects’ efforts is assumed as the estimate. If no project is found in the $\pm 20\%$ range, the range is progressively increased until at least one project in the range is found. The estimation for a given project of size S is thus performed according to the following procedure:

```

R=0.2
P = {}
while |P| < 1 do
  for all Pi in the set of projects
    if (1-R)×S ≤ Pi.size ≤ (1+R)×S
      then P = P ∪ {Pi.effort}
  R=R+0.1
done
EstimatedEffort = mean(P)

```

So, for instance, given P3 (whose size is 171 UFP), P1 and P10 (which have size 185 UFP and 179 UFP, respectively) are selected as analogue projects. In fact, these are the only projects that have size in the range $171 \pm 20\%$, i.e., in range 137–205. Since the developments of the selected projects required 410 and 272 PersonHours, we assume that P3 requires $(410+272)/2 = 341$ PersonHours.

When analogue projects are selected also on the basis of complexity density, the condition for inclusion into the set of analogous projects is modified in order to select projects whose size S_i and complexity density C_i satisfy the condition $((1-R) \times S \leq S_i \leq (1+R) \times S) \wedge ((1-R) \times C \leq C_i \leq (1+R) \times C)$, where C is the complexity density of the project being estimated.

A. Estimation by Analogy using FPA Measures

In this subsection, we present the results of estimation by analogy based on FPA measures: we used UFP as a size measure and Path/UFP as complexity density measure.

UFP were chosen because they are the most obvious size measures, when FPA is used. Similarly, we used CFP and UUCP in the following sections.

Effort estimates and the differences with respect to actual efforts are illustrated in Table IX, where columns labeled “CA est. (2 var)” report results concerning the estimation based on both size and complexity density.

It is easy to see that the estimation error is generally larger for the estimates based only on size similarity, than for the estimates based on the similarity of both size and complexity density.

TABLE IX. ANALOGY-BASED ESTIMATES OBTAINED USING FPA MEASURES

PID	Actual Effort	CA est.	CA est. (2 var)	Error CA	Error CA (2 var)
1	410	459	553	49 (12%)	143 (35%)
2	474	557	392	84 (18%)	-82 (-17%)
3	382	341	410	-41 (-11%)	28 (7%)
4	285	262	263	-23 (-8%)	-22 (-8%)
5	328	252	348	-76 (-23%)	20 (6%)
6	198	282	266	84 (42%)	68 (34%)
7	442	163	198	-279 (-63%)	-244 (-55%)
8	723	341	410	-382 (-53%)	-313 (-43%)
9	392	474	474	82 (21%)	82 (21%)
10	272	505	230	233 (86%)	-43 (-16%)
11	131	301	272	170 (130%)	141 (108%)
12	348	237	328	-111 (-32%)	-20 (-6%)
13	243	273	285	31 (13%)	43 (18%)
14	300	147	147	-153 (-51%)	-153 (-51%)
15	147	169	169	22 (15%)	22 (15%)
16	169	134	121	-35 (-21%)	-48 (-28%)
17	121	239	158	118 (98%)	37 (31%)
18	342	268	268	-74 (-22%)	-74 (-22%)
19	268	342	342	74 (28%)	74 (28%)

TABLE X. ANALOGY-BASED ESTIMATES OBTAINED USING FPA MEASURES: MMRE, MDMRE AND PRED(25)

	CA	CA (2 var)
Mean	39.2%	28.8%
Median	23.3%	21.6%
Pred(25)	52.6%	52.6%

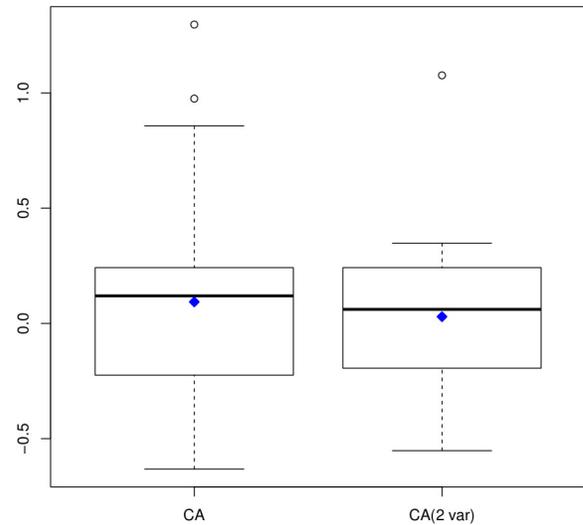


Figure 5. Estimation by analogy based on FPA measures: relative errors.

The relative estimation errors reported in parentheses are computed as the estimation errors divided by the actual efforts, expressed as percentages. For instance, for project 1 $ErrorCa = 49$, $ActualEffort = 410$, thus the relative error is $49/410 = 12\%$.

For eleven projects, considering also the complexity density in the selection of analogous projects leads to smaller absolute relative errors. In five cases, considering also the complexity density does not cause any change in the absolute relative error. Only for three projects (P1, P13 and P16) the relative absolute error is smaller when the estimate is based only on size.

The mean and median absolute relative errors (i.e., MMRE and MDMRE) are reported in Table X, together with Pred(25). Table X shows that considering complexity density allows for better accuracy than considering size alone to identify analogous projects. This fact is confirmed by the distributions of relative errors (illustrated by the boxplot in Fig. 5) and absolute relative errors (illustrated by the boxplot in Fig. 6). Fig. 5 shows that when complexity density is considered, the median, the mean (represented as a diamond) and the errors in general are closer to zero. Fig. 6 shows that the median, the mean and the errors in general are smaller when complexity density is taken into consideration in the EbA.

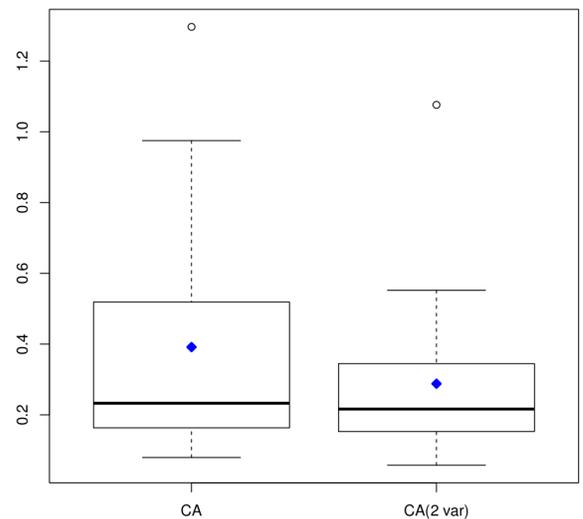


Figure 6. Estimation by analogy based on FPA measures: absolute relative errors.

TABLE XI. ANALOGY-BASED ESTIMATES USING COSMIC MEASURES

PID	Actual Effort	CA est.	CA est. (2 var)	Error CA		Error CA (2 var)	
1	410	598	428	188	(46%)	18	(4%)
2	474	435	312	-39	(-8%)	-161	(-34%)
3	382	458	474	75	(20%)	91	(24%)
4	285	304	357	19	(7%)	72	(25%)
5	328	231	330	-97	(-29%)	2	(1%)
6	198	286	286	88	(44%)	88	(44%)
7	442	249	249	-193	(-44%)	-193	(-44%)
8	723	373	317	-350	(-48%)	-405	(-56%)
9	392	455	483	63	(16%)	91	(23%)
10	272	308	308	36	(13%)	36	(13%)
11	131	249	147	118	(90%)	16	(12%)
12	348	228	323	-120	(-34%)	-25	(-7%)
13	243	485	519	242	(100%)	276	(114%)
14	300	215	338	-85	(-28%)	38	(13%)
15	147	246	126	99	(67%)	-21	(-14%)
16	169	268	268	99	(59%)	99	(59%)
17	121	266	147	145	(120%)	26	(21%)
18	342	266	338	-76	(-22%)	-4	(-1%)
19	268	169	169	-99	(-37%)	-99	(-37%)

B. Estimation by using COSMIC Measures

In this subsection, we present the results of estimation by analogy based on COSMIC measures: we used CFP as a size measure and Path/CFP as complexity density measure.

Effort estimates and the differences with respect to actual efforts are illustrated in Table XI.

For eight projects, considering also the complexity density in the selection of analogous projects leads to smaller absolute relative errors. For six projects the relative absolute error is smaller when the estimate is based only on size analogy. In five cases, considering also the complexity density does not cause any change in the absolute relative error.

MMRE, MdMRE and Pred(25) are given in Table XII. Table XII shows that considering complexity density allows for better accuracy than considering size alone to identify analogous projects. This fact is confirmed by the distributions of relative errors and absolute relative errors (illustrated by the boxplots in Fig. 7 and Fig. 8, respectively).

TABLE XII. ANALOGY-BASED ESTIMATES OBTAINED USING COSMIC MEASURES: MMRE, MdMRE AND PRED(25)

	CA	CA (2 var)
MMRE	43.8%	28.8%
MdMRE	36.9%	23.1%
Pred(25)	31.6%	57.9%

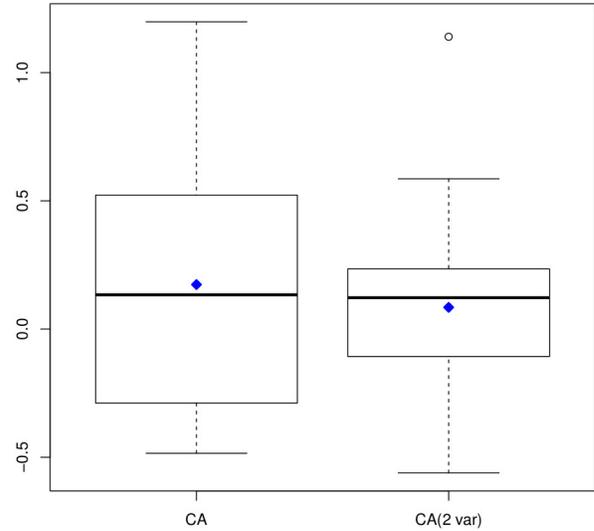


Figure 7. Estimation by analogy based on COSMIC measures: relative errors.

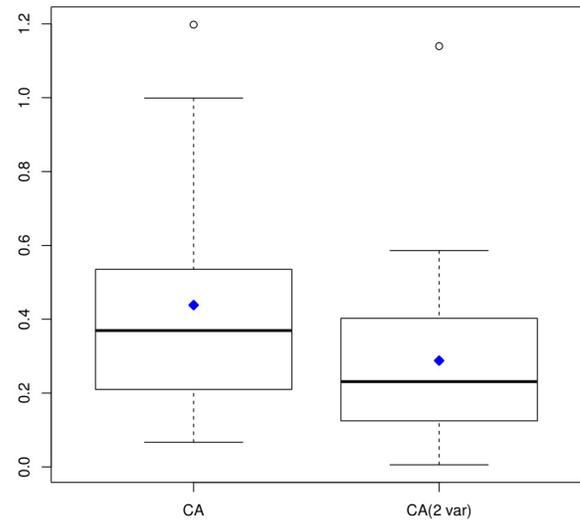


Figure 8. Estimation by analogy based on COSMIC measures: absolute relative errors.

Fig. 7 shows that when complexity density is considered, the median, the mean (represented as a diamond) and the errors in general are (slightly) closer to zero. Fig. 8 shows that the median, the mean and the errors in general are smaller when complexity density is taken into consideration in the EbA.

We can also observe that the results obtained when using COSMIC measures are very similar to those obtained using FPA measures.

C. Estimation by Analogy using Use Case Measures

In this subsection, we present the results of estimation by analogy based on Use Case measures: we used UUCP as a size measure and Path/UUCP as complexity density measure.

Effort estimates and the differences with respect to actual efforts are illustrated in Table XIII.

TABLE XIII. ANALOGY-BASED ESTIMATES OBTAINED USING USE CASE MEASURES

PID	Actual Effort	CA est.	CA est. (2 var)	Error CA	Error CA (2 var)
1	410	723	474	313 (76%)	64 (15%)
2	474	723	723	249 (53%)	249 (53%)
3	382	296	323	-86 (-23%)	-59 (-15%)
4	285	305	341	20 (7%)	56 (20%)
5	328	298	273	-30 (-9%)	-55 (-17%)
6	198	263	314	65 (33%)	116 (59%)
7	442	282	291	-160 (-36%)	-151 (-34%)
8	723	442	474	-281 (-39%)	-249 (-34%)
9	392	257	243	-135 (-34%)	-150 (-38%)
10	272	317	401	45 (17%)	129 (47%)
11	131	317	268	186 (142%)	137 (105%)
12	348	295	328	-53 (-15%)	-20 (-6%)
13	243	387	387	145 (60%)	145 (60%)
14	300	201	270	-99 (-33%)	-30 (-10%)
15	147	246	195	99 (67%)	48 (32%)
16	169	305	305	136 (80%)	136 (80%)
17	121	251	208	130 (107%)	87 (71%)
18	342	209	210	-133 (-39%)	-132 (-38%)
19	268	227	147	-41 (-15%)	-121 (-45%)

For ten projects, considering also the complexity density in the selection of analogous projects leads to smaller absolute relative errors. For six projects the relative absolute error is smaller when the estimate is based only on size analogy.

The MMRE, MdMRE and Pred(25) are reported in Table XIV. Table XIV shows that considering complexity density allows for better accuracy than considering size alone to identify analogous projects only as far as is concerned.

The distributions of relative errors (illustrated by the boxplot in Fig. 9) and absolute relative errors (illustrated by the boxplot in Fig. 10) show that taking into account complexity density when selecting analogue projects causes marginal improvements in estimation accuracy.

TABLE XIV. ANALOGY-BASED ESTIMATES OBTAINED USING USE CASE MEASURES: MMRE, MdMRE AND PRED(25)

	CA	CA (2 var)
MMRE	46.6%	41.1%
MdMRE	36.3%	38.1%
Pred(25)	31.6%	31.6%

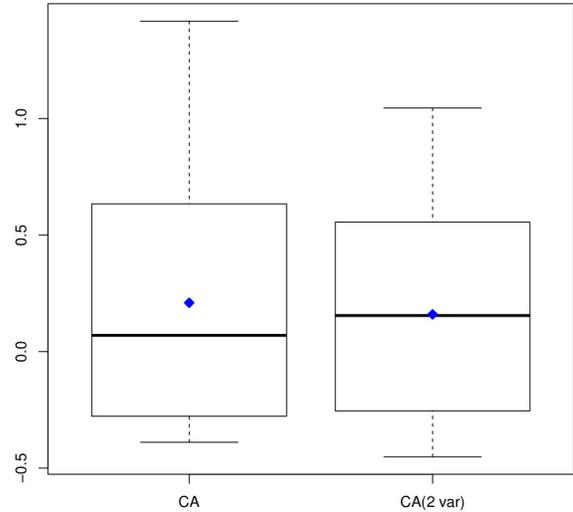


Figure 9. Estimation by analogy based on Use Case measures: relative errors.

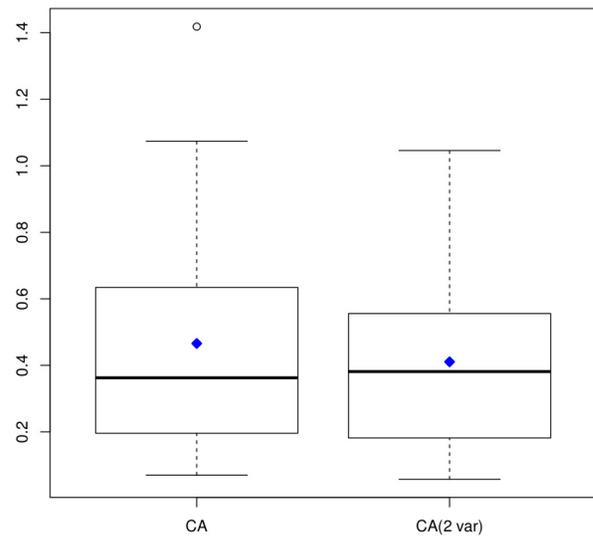


Figure 10. Estimation by analogy based on Use Case measures

As a final observation, we note that using both size and complexity density in the selection of analogue projects generally results in more accurate estimates. Only when the size measures are based on use cases, considering complexity density does not lead to a clear improvement in estimation accuracy.

We can conclude that complexity density appears to be a relevant factor to be considered when EbA is adopted.

VII. DISCUSSION

Table XV summarizes the accuracy of the effort models described in Section V. These models are all those statistically significant and featuring $R^2 > 0.6$.

TABLE XV. ACCURACY OF EFFORT MODELS

Model	MMRE	Pred(25)	Error range
Effort = $-29.9 + 139.1 \times$ Path/FPr	34%	63%	-46%–162%
Effort = $120.6 \times \text{FPATrans}^{0.654} \times$ (Path/UFP) ^{0.958}	71%	53%	-40%–543%
Effort = $12.2 \times \text{FPATrans}^{0.816} \times$ (Path/FPATrans) ^{0.976}	25%	63%	-60%–108%
Effort = $112.2 \times \text{CFP}^{0.391} \times$ (Path/CFP) ^{1.298}	22%	68%	-30%–77%
Effort = $231.8 \times \text{FPr}^{0.377} \times$ (Path/CFP) ^{1.468}	23%	68%	-27%–97%
Effort = $21.9 \times \text{UC}^{0.68} \times$ (Path/UC) ^{0.728}	24%	63%	-45%–73%
Effort = $24.9 \times \text{UUCP}^{0.679} \times$ (Path/UUCP) ^{0.775}	24%	63%	-45%–74%

It is easy to see that all models in Table XV have at least one parameter that accounts for functional complexity per size unit.

It should be noted that we found also a model based *exclusively* on complexity density. This model was rather unexpected, as it says that the size of the programs is not important at all. This result is probably due to the fact that the variation of size is relatively little in the set of projects that we analysed. Additional research is needed to explore this point.

Having shown that functional complexity per size unit is essential for regression models, we looked at the role that functional complexity per size unit can play in EbA. To this end, EbA was applied using two criteria for determining analogue projects:

- According to size only;
- According to both size and functional complexity per size unit.

Table XVI illustrates the results of EbA concerning the accuracy of estimates via MMRE, MdMRE and Pred(25).

TABLE XVI. ACCURACY OF ANALOGY-BASED ESTIMATES

Variables used to identify analogue projects	MMRE	MdMRE	Pred(25)
UFP	39.2%	23.3%	52.6%
UFP, Path/UFP	28.8%	21.6%	52.6%
CFP	43.8%	36.9%	31.6%
CFP, Path/CFP	28.8%	23.1%	57.9%
UUCP	46.6%	36.3%	31.6%
UUCP, Path/UUCP	41.1%	38.1%	31.6%

Table XVI confirms the relevance of functional complexity per size unit, as it helps increasing accuracy. Actually it can be noted that EbA's accuracy is generally worse than regression models'. However, using functional complexity per size unit for determining analogues projects tends to make estimation accuracy closer to regression models'.

VIII. THREATS TO VALIDITY

As in any software engineering empirical study, several issues threaten the validity of the results. Here we discuss such factors and the actions that have been undertaken to mitigate them.

The limited size of the available dataset can be regarded as a first threat to internal validity. The used dataset is sufficiently large to support statistically significant analysis; however, the fact that our dataset is representative of most software systems is doubtful. To this end, we note that the projects from which the data used in the paper were derived are all real development projects, as those in best known datasets, like the PROMISE [39] and ISBSG [22] datasets. Some of our projects are rather small, but the majority of our projects (namely 11 out of 19) have size in the 100–340 UFP, thus they can be considered medium-sized projects (40% of the projects in the ISBSG dataset are in the 100–340 UFP range, while more than half have size smaller than 340 UFP).

Another possible threat to the validity of the study derives from part of the projects being academic projects. However, the projects that were carried out at the Austral University were developed using techniques, tools and methodologies similar to those used in the industrial projects. Accordingly, we do not expect that these projects required substantially different effort than other projects.

IX. CONCLUSION

The work reported here moves from the consideration that development effort depends (also) on the complexity or the amount of computation required, but no suitable measure has emerged as a reliable way for capturing such complexity. In fact, very popular methods like COCOMO II [11][15] still use just an ordinal scale measure for complexity, based on the subjective evaluation performed by the user.

We approached the problem of measuring the required functional complexity by considering the most relevant approaches presented in the literature, and testing them on a set of projects that were measured according to FPA, COSMIC and Use Case-based functional size measurement methods.

The results of our analysis do not allow us to draw definite conclusions, since our observations are based on a specific set of data (see Table I). However, we observed that all the models obtained were based on a notion of computation density, which is based on the measure of Paths [43], i.e., the number of distinct computation flows in functional processes. Similarly, Estimation by Analogy appears to benefit from the possibility of using the notion of computation (or complexity) density in identifying analogue projects.

Since Paths are quite easy to measure [33] and appear as good effort predictors, we suggest that future research on effort estimation takes into consideration the possibility of involving a Path based measure of functional complexity.

An important results for practitioners is that functional complexity appears as a factor that affects development effort; accordingly, whatever method is used for effort

estimation, it is advisable to take functional complexity into due consideration.

We plan to continue experimenting with measures of functional complexity. Since in this type of experimentations a critical point is the difficulty to get measures, we kindly invite all interested readers that are involved in effort estimations to perform functional complexity measurement and share the data with us and the research community.

ACKNOWLEDGMENT

The research presented in this paper has been partially funded by the project “Metodi, tecniche e strumenti per l’analisi, l’implementazione e la valutazione di sistemi software” funded by the Università degli Studi dell’Insubria, and by the Research Fund of the School of Engineering of Austral University.

REFERENCES

- [1] L. Lavazza and G. Robiolo, “Functional Complexity Measurement: Proposals and Evaluations,” Proc. 6th Int. Conf. on Software Engineering Advances (ICSEA 2011).
- [2] A. Albrecht, “Measuring Application Development Productivity,” Proc. IBM Application Development Symp. I.B.M. Press, 1979.
- [3] A.J. Albrecht and J.E. Gaffney, “Software Function, Lines of Code and Development Effort Prediction: a Software Science Validation,” IEEE Transactions on Software Engineering, vol. 9, November 1983.
- [4] N. Aggarwal, N. Prakash, S. Sofat, “Web Hypermedia Content Management System Effort Estimation Model,” SIGSOFT Software Engineering Notes, vol. 34, March 2009.
- [5] M. AlSharif, W.P. Bond, and T. Al-Otaiby, “Assessing the complexity of software architecture,” Proc 42nd annual Southeast regional conference (ACM-SE 42). ACM, New York, NY, USA, 2004, pp. 98-103.
- [6] B. Anda, E. Angelvik, and K. Ribu, “Improving Estimation Practices by Applying Use Case Models,” Lecture Notes In Computer Science, vol. 2559, Springer, 2002, pp. 383-397.
- [7] L. Baresi, S. Colazzo, L. Mainetti, and S. Morasca, “W2000: A modeling notation for complex Web applications,” In Web Engineering, E. Mendes and N. Mosley. Eds., Springer-Verlag, 2006.
- [8] L. Baresi, and S. Morasca, “Three Empirical Studies on Estimating the Design Effort of Web Applications,” ACM Transactions on Software Engineering and Methodology, vol. 16, September 2007.
- [9] H. Bashir, and V. Thomson, “Models for Estimating Design Effort and Time,” Design Studies, vol. 22, March, Elsevier, 2001.
- [10] B. Bernárdez, A. Durán, and M. Genero, “Empirical Evaluation and Review of a Metrics-Based Approach for Use Case Verification,” Journal of Research and Practice in Information Technology, vol. 36, November 2004.
- [11] B.W. Boehm, E. Horowitz, R. Madachy, D. Reifer, B.K. Clark, B. Steece, A. Winsor Brown, S. Chulani and C. Abts, Software Cost Estimation with Cocomo II. Prentice Hall, 2000.
- [12] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide, Addison Weley, 1998.
- [13] L. Briand, S. Morasca, and V.R. Basili, “Property-Based Software Engineering Measurement,” IEEE Transactions on Software Engineering, vol. 22, Month, 1996.
- [14] L. Briand and J. Wust, “Modeling Development Effort in Object-Oriented Systems Using Design Properties,” IEEE Transactions on Software Engineering, vol. 27, November 2001.
- [15] COCOMO II Model Definition Manual. http://csse.usc.edu/csse/research/COCOMOII/cocomo_downloads.htm, last access Dec. 20, 2012.
- [16] COSMIC – Common Software Measurement International Consortium, 2009. The COSMIC Functional Size Measurement Method - version 3.0.1 Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2003), May 2009.
- [17] N.E. Fenton, Software Metrics: A Rigorous Approach. Chapman and Hall, London, 1991.
- [18] D.D. Galorath and M.W. Evans, Software Sizing, Estimation, and Risk Management, Auerbach Publications, 2006.
- [19] C. Gencel and O. Demirors, “Functional Size Measurement Revisited,” ACM Transactions on Software Engineering and Methodology, vol. 17, June 2008.
- [20] S. Gupta, G. Sikka, and H. Verma, “Recent methods for software effort estimation by analogy,” SIGSOFT Softw. Eng. Notes , vol. 36, August 2011, pp 1-5.
- [21] T. Hastings and A. Sajeve, “A Vector-Based Approach to Software Size Measurement and Effort Estimation,” IEEE Transactions on Software Engineering, vol. 27 April 2001.
- [22] International Software Benchmarking Standards Group: Worldwide Software Development: The Benchmark, release 11, 2009.
- [23] ISO/IEC19761:2003, Software Engineering – COSMIC-FFP – A Functional Size Measurement Method, International Organization for Standardization, Geneve, 2003.
- [24] ISO/IEC 20926: 2003, Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting Practices Manual, International Organization for Standardization, Geneve, 2003.
- [25] M. Jørgensen, M. Boehm, S. Rifkin, “Software Development Effort Estimation: Formal Models or Expert Judgment?,” IEEE Software, vol. 26, March-April 2009.
- [26] C. Jones, A Short History of Function Points and Feature Points. Software Productivity Research, Inc., Burlington, Mass., 1986.
- [27] C. Jones, Strengths and Weaknesses of Software Metrics. Version 5, Software Productivity Research, 2006.
- [28] G. Karner, Resource Estimation for Objectory Projects. Objectory Systems, 1993.
- [29] G. Karner, “Metrics for Objectory,” Diploma thesis, University of Linköping, 1993.
- [30] B. Kitchenham, S.L. Pfleeger, B. McColl, and S. Eagan, “An Empirical Study of Maintenance and Development Accuracy,” Journal of Systems and Software, vol. 64, October 2002.
- [31] B. Kitchenham, L.M. Pickard, S.G. MacDonell, M.J. Shepperd, “What accuracy statistics really measure [software estimation],” IEE Proceedings - Software vol. 148 , June 2001, pp. 81 – 85.
- [32] S. Kusumoto, F. Matukawa, K. Inoue, S .Hanabusa, and Y. Maegawa, “Estimating effort by Use Case Points: Method, Tool and Case Study,” Proc. 10th International Symposium on Software Metrics, 2004.
- [33] L. Lavazza and G. Robiolo, “Introducing the Evaluation of Complexity in Functional Size Measurement: a UML-based Approach,” Proc. 4th Int. Symposium on Empirical Software Engineering and Measurement (ESEM 2010).
- [34] L. Lavazza and G. Robiolo, “The Role of the Measure of Functional Complexity in Effort Estimation,” Proc. 6th Int. Conf. on Predictive Models in Software Engineering (PROMISE 2010).

- [35] G. Levesque V. Bevo, and Tran Cao, D., "Estimating Software size with UML Models," Proc. 2008 C3S2E conference, ACM International Conference Proceeding Series, vol. 290, 2008.
- [36] K. Lind and R. Heldal, "Categorization of Real-time Software Components for Code Size Estimation," Proc. ACM-IEEE Int. Symp. on Empirical Software Engineering and Measurement (ESEM 2010). ACM, New York, NY, USA, 2010.
- [37] T.J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, vol.2, December 1976.
- [38] E. Mendes, N. Mosley, and S. Counsell, "A Comparison of Length, Complexity and Functionality as Size Measures for Predicting Web Design and Authoring Effort," Proc. Evaluation and Assessment in Software Engineering Conference (EASE 2001).
- [39] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The PROMISE Repository of Empirical Software Engineering Data <http://promisedata.googlecode.com>," West Virginia University, Department of Computer Science, 2012, last access Dec. 20, 2012.
- [40] S. Misra, "Modified Cognitive Complexity Measure," Proc. 21st international conference on Computer and Information Sciences (ISCIS'06), A. Levi, E. Savaş, H. Yenigün, S. Balcisoy and Y. Saygin Eds., Springer-Verlag, Berlin, Heidelberg, 2006, pp. 1050-1059.
- [41] N. Mittas and L. Angelis, "Combining Regression and Estimation by Analogy in a Semi-parametric Model for Software Cost Estimation," Proc. 2nd ACM-IEEE Int. Symp. on Empirical Software Engineering and Measurement (ESEM 2008). ACM, New York, NY, USA, 2008, pp. 70-79.
- [42] S. Morasca, "On the Use of Weighted Sums in the Definition of Measures," ICSE Workshop on Emerging Trends in Software Metrics (WETSOM 2010).
- [43] G. Robiolo, G. and R. Orosco, "Employing Use Cases to Early Estimate Effort with Simpler Metrics," Innovations Syst. Softw. Eng, vol.4, April 2008.
- [44] S. Sengupta, A. Kanjilal, and S. Bhattacharya, "Measuring Complexity of Component Based Architecture: a Graph Based Approach," SIGSOFT Softw. Eng. Notes vol. 36, January 2011, pp. 1-10.
- [45] M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies", IEEE Transactions on Software Engineering, vol. 23, month11, 1997, pp. 736-774.
- [46] D. Tran Cao, G. Lévesque, and A. Abran, "From Measurement of Software Functional Size to Measurement of Complexity", Int. Conf. on Software Maintenance (ICSM 2002).
- [47] D. Tran Cao, G. Lévesque, and J-G. Meunier, "A Field Study of Software Functional Complexity Measurement", Proc. 14th Int. Workshop on Software Measurement (IWSM/METRIKON 2004).
- [48] G. Visaggio, "Structural Information as a Quality Metric in Software Systems Organization," Proc. Int. Conf. on Software Maintenance, 1997, pp. 92-99.
- [49] T. Wijayasiriwardhane and R. Lai, "Component Point: A System-level Size Measure for Component-Based Software Systems," Journal of Systems and Software vol. 83, month 2010, pp. 2456-2470.
- [50] A. Whitmire, "An Introduction to 3D Function Points," Software Development, vol. 3, April 1995.
- [51] Y. Yavari, M. Afsharchi, and M. Karami, "Software Complexity Level Determination Using Software Effort Estimation Use Case Points Metrics," 5th Malaysian Conference in Software Engineering (MySEC 2011), pp. 257-262.

Metrics and Measurements in Global Software Development

Maarit Tihinen and Päivi
Parviainen

Digital Service Research
VTT Technical Research Centre of
Finland
maarit.tihinen@vtt.fi
paivi.parviainen@vtt.fi

Rob Kommeren

Digital Systems & Technology
Philips,
The Netherlands
r.c.kommeren@philips.com

Jim Rotherham

Project Management Office
Symbio,
Finland
jim.rotherham@symbio.com

Abstract—Today products are increasingly developed globally in collaboration between subcontractors, third-party suppliers and in-house developers. However, management of a distributed product development project is proven to be more challenging and complicated than traditional single-site development. From the viewpoint of project management, the measurements and metrics are important elements for successful product development. This paper is focused on describing a set of essential metrics that are successfully used in Global Software Development (GSD). In addition, visualised examples are given demonstrating various industrial experiences of use. Even if most of the essential metrics are similar as in single-site development, their collection and interpretation need to take into account the GSD aspects. One of the most important reasons for choosing proposed metrics was their provision of early warning signs - to proactively react to potential issues in the project. This is especially important in distributed projects, where tracking the project status is needed and more complex. In this paper, the first ideas of GSD specific metrics are presented based on the common challenges in GSD practice.

Keywords-metrics; measurements; global software development; distributed product development

I. INTRODUCTION

Global Software Development (GSD) is increasingly common practice in industry due to the expected benefits, such as lower costs and utilising resources globally. GSD brings several additional challenges to the development, which also affects the measurement practices, results and metrics interpretation. A current literature study showed that there is little research on GSD metrics or experiences of their use. This paper is enhanced and extended version of the ICSEA 2011 conference paper “Metrics in distributed product development” [1] where the metrics set had been successfully used in GSD were introduced. In this paper, the published metric set (with an example set of visualised metrics) was given with industrial experiences of their use. In addition, challenges faced during GSD are discussed from the viewpoint of metrics and measurements as well as potential GSD specific metrics.

Software metric is a valuable factor for the management and control of many software related activities, for example; cost, effort and schedule estimation, productivity, reliability and quality measures. Traditionally software measurement has been understood as an information gathering process. For example, software measurement is defined by [2] as follows: “The software measurements is the continuous process of defining, collecting and analysing data on the software development process and its products in order to understand and control the process and its products and to supply meaningful information to improve that process and its products”. The measurement data item consists of numeric data (e.g., efforts, schedules) or a pre-classified set of categories (e.g., severity of defects: minor, medium, major). Software metrics can consist of several measurement data items singly or in combination. Metric visualisation is a visual representation of the collected and processed information about software systems. Typically software metrics are visualised for presenting this information in a meaningful way that can be understood quickly. For example, visualising metrics through charts or graphs is usually easier to understand than long textual or numerical descriptions.

The main purpose of measurements and metrics in software production is to create the means for monitoring and controlling which provide support for decision-making and project management [3]. Traditionally, the software metrics are divided into process, product and resource metrics [4]. In the comprehensive measurement program, all these dimensions should be taken into consideration while interpreting measurement results; otherwise the interpretation may lead to wrong decisions or incorrect actions. A successful measurement program can prove to be an effective tool for keeping on top of the development effort, especially for large distributed projects [5]. However, many problems and challenges have been identified that reduce and may even eliminate all interests to the measurements. For example, not enough time is allocated for the measurement activities during a project, or not enough visible benefits are gained by the project doing the measurement work (e.g., data is useful only at the end of

project, not during the project). In addition, the “metric enthusiasts” may define too many metrics making it too time consuming to collect and analyse the data. Thus, it’s beneficial [5] to define core metrics to collect across all projects to provide benchmarking data for projects, and to focus on measurements that come naturally out of existing practices and tools.

GDS development enables product development to take place independently of the geographical location, individuals or organizations. In fact, today the products are increasingly developed globally in collaboration between subcontractors, third party suppliers and in-house developers [6]. In practice distributed projects struggle with the same problems as single-site projects including problems related to managing quality, schedule and cost. Distribution only makes it even harder to handle and control these problems [7][8][9][10][11]. These challenges are caused by various issues, for example, less communication – especially informal communication – caused by distance between partners, and differences in background knowledge of the partners. That’s why, in distributed projects the systematic monitoring and reporting of the project work is especially important, and measurement and metrics are an important means to do that effectively.

Management of a distributed product development project is more challenging than traditional development [12]. Based on an industrial survey [13], one of the most important topics in the project management in distributed software development is detailed project planning and control during the project. In GSD, this includes; dividing work by sites into sub-projects, clearly defined responsibilities, dependencies and timetables, along with regular meetings and status monitoring.

In this paper, a set of essential metrics used in GSD is discussed with experiences of their use. The main purpose is to introduce the selected metric set from the viewpoint of their proactive role in decision-making during globally distributed software development. The chosen metrics indicate a well-rounded view of status in the various engineering disciplines and highlight potential issues in the project. This creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. This is especially important in GSD, where information of project status is not readily available but requires special effort, distributed over sites and companies.

The amount of the metrics is intentionally kept as limited as possible. Also, the metrics should be such, that they provide online information during the projects, in order to enable fast reaction to potential problems during the project. The metrics and experience presented in the paper are based on metrics programs of two companies, Philips and Symbio. Royal Philips Electronics is a global company providing healthcare, consumer lifestyle and lighting products and services. Digital Systems & Technology is a unit within Philips Research that develops first-of-a-kind products in the area of healthcare, well-being and lifestyle. The projects follow a defined process and are usually distributed over sites and/or use subcontractors as part of product development. Symbio Services Oy provides tailored

services to organizations seeking to build tomorrow's technologies. Well-versed in a variety of software development methodologies and testing best practices, Symbio's specialized approaches and proprietary processes begin with product design and continue through globalization, maintenance and support. Symbio has built a team of worldwide specialists that focus on critical areas of the product development lifecycle. Currently, Symbio employs around 1400 people and their project execution is distributed between sites in the US, Sweden, Finland and China.

The metrics and discussion in the paper is based on GSD improvement work carried out during several years, in several research projects, including experiences from 54 industrial cases (see Parviainen [14], SameRoomSpirit Wiki [15]). This paper focuses especially on the experiences of two companies, Philips and Symbio.

The paper is structured as follows. Firstly, an overview of related work – available literature and its limitations related to measurements and metrics in distributed product development. This is introduced in Section II. In Section III, basic GSD circumstances with challenges are presented in order to explain the special requirements for measurements in GSD where the proposed metrics set is to be collected and utilised. In Section IV, measurement and metrics background and used terminology are introduced. In Section V, proposed metrics are presented using Rational Unified Process (RUP) [16] approach as a framework. The proposed metric set is presented with visualised examples and industrial experiences of their use. Furthermore, some GSD specific metrics are introduced in Section VI. Finally, discussion about metrics and their experiences is presented in Section VII and the conclusions are discussed in Section VIII.

II. RELATED WORK

There are several papers that discuss globally distributed software engineering and its challenges, for example, [5], [17] and [18]. Also, metrics in general and for specific aspects have been discussed in numerous papers and books for decades. However, little GSD literature has focused on metrics and measurements or even discusses the topic. Da Silva et al. [12] report similar conclusion based on analysis of distributed software development (DSD) literature published during 1999 – 2009: they state as one of their key findings that the “vast majority of the reported studies show only qualitative data about the effect of best practices, models, and tools on solving the challenges of DSD project management. In other words, our findings indicate that strong (quantitative) evidence about the effect of using best practices, models, and tools in DSD projects is still scarce in the literature.” Bourgault et al. [19] reported similar findings, “Clearly, research into distributed projects’ performance metrics and measurement needs more attention from researchers and practitioners so that it can contribute to the development and diffusion of well-designed management information systems.”

The papers that have discussed some metrics for GSD usually focus on some specific aspect, for example, Korhonen and Salo [18], discuss quality metrics to support

defect management process in a multi-site organization. Misra [20] presents a cognitive weight complexity metric (CWCM) for unit testing in a global software development environment. Lotlikar et al. [21] propose a framework for global project management and governance including some metrics with the main goal to support work allocation to various sites. Lane and Agerfalk [22] use another framework as an analytic device to investigate various projects performed by distributed teams in order to explore further the mechanisms used in industry both to overcome obstacles posed by distance and process challenges and also to exploit potential benefits enabled by GDS. Similarly, Piri and Niinimäki [23] applied Word Design Questionnaire (WDQ) that consists of total of 21 sum variables in four categories (task characteristics, knowledge characteristics, social characteristics, and work context) to compare differences between the co-located and the distributed projects by metrics - “work design”, “team dynamics”, “teamwork quality”, “project performance” and “individual satisfaction”. These kinds of frameworks could be used to evaluate effectiveness of distributed team configuration during GSD projects as well. Peixoto et al. [17] discuss effort estimation in GSD, and one of their conclusions is that “GSD projects are using all kinds of estimation techniques and none of them is being consider as proper to be used in all cases that it has been used”, meaning, that there is no established technique for GSD projects. In addition, some effort has also been invested in defining how to measure success of GSD projects [24], and these metrics mainly focus on cost related metrics and are done after project completion. These papers usually use common metrics that are not specific for GSD projects. For example, Ramasubbu and Balan [25] use 11 metrics (productivity, quality, dispersion, prevention QMA (Quality Management Approach), appraisal QMA, failure QMA, code size, team size, design rework, upfront investment and reuse), development productivity and conformance quality to evaluate how work dispersion effects to identified metrics. However, these metrics have not been used to gather information, indicators or experiences from ongoing distributed development.

Furthermore, only few papers discuss measurement tooling for GSD projects. Simmons [26] describes a PAMPA tool, where an intelligent agent tracks cost driver dominators to determine if a project may fail and tells managers how to modify project plans to reduce probability of project failure. Additionally, Simmons and Ma [27] discuss a software engineering expert system (SEES) tool where the software professional can gather metrics from CASE tool databases to reconstruct all activities in a software project from project initiation to project termination. Da Silva et al [28] discuss software cockpits from GSD viewpoint. They propose to examine various visualizations in the context of software cockpits, at-a-glance computer controlled displays of development-related data collected from multiple sources. They present three visualizations: (1) shows high-level information about teams and dependencies among them in an interactive world map, (2) displays the system design through a self-updating view of the current state of the software implementation, and (3) is a 3D visualization that

presents an overview of current and past activities in individual workspaces.

The focus of this paper is to introduce a metrics set that creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. Furthermore, the paper gives several visualised examples of metrics that can be utilised while monitoring on-going GSD projects. The introduced metrics set can be seen as ‘balanced score card’, on which management can balance insights (~status) from time, effort, cost, functionality (requirements) and quality (tests) perspective.

III. BASIC GSD CIRCUMSTANCES WITH CHALLENGES

Parviainen [14] describes problems and challenges that are directly caused by the basic GSD circumstances. These challenges influence measurements and metrics and their interpretation during distributed software development. These challenges are mainly an intrinsic and natural part of GSD and they can either complicate globally distributed product development or even cause further challenges. The basic circumstances are:

- Multiple parties, meaning two or more different teams and sites (locations) of a company or different companies.
- Time difference and distance that are caused by the geographical distribution of the parties.

Problems caused by these circumstances include; issues such as unclear roles and responsibilities for the different stakeholders in different parties or locations, knowing the contact persons (e.g., responsibilities, authorities and knowledge) from different locations and establishing and ensuring a common understanding across distance. The basic GSD circumstances can also lead to poor transparency and control of remote activities as well as difficulties in managing dependencies over distance, problems in coordination and control of the distributed work and integration problems, for example. Problems may also be caused by basic circumstances in terms of accessing remote databases and tools or accordingly they may generate data transfer problems caused by the various data formats between the tools or different versions of the tools used by the different teams. The basic circumstances may also cause problems with data security and access to databases or another organisation's resources.

A commonly referenced classification for challenges caused by GSD is [29][30]:

- Communication breakdown (loss of communication richness)
- Coordination breakdown
- Control breakdown (geographical dispersion)
- Cohesion barriers (loss of “teamness”)
- Culture clash (cultural differences).

Communication breakdown (loss of communication richness). Human beings communicate best when they are communicating face-to-face. In GSD, face-to-face communication decreases due to distance, causing misunderstandings and lack of information over sites. For example, communication over distance can lead to

misinterpretation because people cannot communicate well due to language barriers.

Coordination breakdown. Software development is a complex process that requires on-going adjustments and coordination of shared tasks. In geographically distributed projects, the small adjustments usually made in face-to-face contact do not take place or it is not easy to make adjustments. This can cause problem solving to be delayed or the project to go down the wrong track until it becomes very expensive to fix. GSD also sets additional requirements for planning, for example, the need for coordination between teams and the procedures and contacts for how to work with partners needs to be defined [31][32][33]. Coordination breakdown can also cause a number of specific problems; for example, Battin et al. [34] reported a number of software integration problems, which were due to a large number of independent teams. Wahyudin et al. [35] state that GSD demands more from project management. In addition to the project managers, the project members such as testers, technical leaders, and developers also need to be kept informed and notified of certain information and events that are relevant to their roles' objectives in timely manner which provides the conditions for in-time decision making.

Control breakdown (geographical dispersion). GSD means that management by walking around the development team is not feasible and, instead, telephones, email and other communication means (e.g., chat servers) must be used. These types of communication tools could be consider as less effective - not always providing a clear and correct status of the development site. Also, dividing the tasks and work across development sites, and managing the dependencies between sites is difficult due to the restraints of the available resources, the level of expertise and the infrastructure [34][36][37]. According to Holmstrom et al. [38], creating the overlap in time between different sites is challenging despite the flexible working hours and communication technologies that enable asynchronous communication. Lack of overlap leads to a delay in responses with a feeling of "being behind", "missing out" and even losing track of the overall work process.

Cohesion barriers (loss of "teamness"). In working groups that are composed of dispersed individuals, the team is unlikely to form tight social bonds, which are a key to a project's success. Lack of informal communication, different processes and practices have a negative impact on teamness [31][32][34]. Furthermore, fear (e.g., of losing one's job to the other site) has direct negative impact on trust, team building co-operation and knowledge transfer, even where good relationships existed beforehand. According to Casey and Richardson [39] fear and lack of trust negatively impact the building of effective distributed teams, resulting in clear examples of not wanting to cooperate and share knowledge with remote colleagues. Al-Ani and Redmiles [40] discuss the role that the existing tools can play in developing trust and providing insights on how future tools can be designed to promote trust. They found that tools can promote trust by sharing information derived from each developer's activities and their interdependencies, leading to a greater likelihood

that team members will rely on each other which leads to a more effective collaboration.

Culture clash (cultural differences). Each culture has different communication norms. In any cross-cultural communication the receiver is more likely to misinterpret messages or cues. Hence, miscommunication across cultures is usually present. Borchers [41] discusses observations of how cultural differences impacted the software engineering techniques used in the case projects. The cultural indexes, power distance (degree of inequality of managers vs. subordinates), uncertainty avoidance (tolerance for uncertainty about the future) and individualism (strength of the relationship between an individual and their societal group), discussed by Hofstede [42], were found to be relevant from the software engineering viewpoint. Holmstrom et al. [38] discuss the challenge of creating a mutual understanding between people from different backgrounds. They concluded that often general understanding in terms of English was good, but more subtle issues, such as political or religious values, caused misunderstandings and conflicts during projects.

IV. MEASUREMENT BACKGROUND

In this section measurement background, the used terminology and traditional measurement methods, with GSD related challenges are introduced.

A. Traditional Metrics and Project Characteristics

Software measurements and metrics have been discussed since 1960's. The metrics have been classified many different ways. For example, they can be divided into basic and additional metrics [43] where basic metrics are size, effort, schedule and defects, and the additional metrics are typically metrics that are calculated or annexed from basic metrics (productivity = software size per used effort). The metrics can also be divided into objective or subjective metrics [43]. The objective metrics are easily quantified and measured, examples including size and effort, while the subjective metrics include less quantifiable data such as quality attitudes (excellent, good, fair, poor). An example of the subjective metrics is customer satisfaction. Furthermore, software metrics can be classified according to the measurement target, product, processes and resources [4]. Example metrics of product entities are size, complexity, reusability and maintainability. Example metrics of process entities are effort, time, number of requirements changes, number of specification/coding faults found and cost. Furthermore, examples of resource entities are age, price, size, maturity, standardization certification, memory size or reliability. These classifications, various viewpoints and the amount of examples merely prove how difficult the selection of metrics really can be during the project.

In addition to different ways of metrics classification, development projects can also be classified. Typically, the project classification is used as a baseline for further interpretation of the metrics and measurements. For example, all kind of predictions or comparison should be done within the same kind of development projects, or the differences should be taken into account. Traditional project

characteristics are, for example; size and duration of a project, type of a project (development, maintenance, operational lifetime, etc.), project position (contractor, subcontractor, internal development etc.), type of software (hardware-related software development, application software, etc.) or used software development approaches (agile, open source, scrum, spiral-model, test driven development, model-driven development, V-model, waterfall model etc.). Furthermore, different phases of development projects have to be taken consideration while analysing gathered measurement data.

B. Traditional Software Measurement and GSD

One of the most commonly used measurement methods at the end of 1990 and the beginning of 2000 was the Goal /Question /Metric (GQM) method. The GQM paradigm [3] represented a systematic approach for tailoring and integrating the objectives of an organisation into measurement goals and their step-wise refinement into measurable values. The GQM method was commonly known and was often used for searching and identifying organisations' strengths and weaknesses relating to the identified improvement goals. Furthermore, several assessment methods, for example CMMI [44] and SPICE (Software Process Improvement and Capability Determination, further known as a standard ISO/IEC 15504 Information technology — Process assessment), were generally used for identifying possible improvements areas and gaining knowledge of the software process of an organisation. In fact, the most of traditional measurements methods were based on expressions of the famous Shewhart cycle, called also the Deming cycle: PDCA (Plan–Do–Check–Act) [45]. The PDCA circle is an iterative four-step management method that is used in business for the control and continuous improvement of processes and products. The traditional methods used in software measurements were generally based on clearly defined and largely stable processes that could be adjusted and improved. In those cases, the improvement actions were mainly done afterwards, for example, in the next project.

In GSD environment, where project stakeholders, work practices and development tools can vary by projects and partners, traditional measurement methods and actions are not adequate if they are used for process improvement purposes. There is little sense, if measurements only prove after the project what has happened during the project, because then it is too late to correct the situation. Furthermore, the lessons learned may not be suitable in the next projects. Overly large measurement programs with time consuming assessments are not worth paying the effort in dynamic GSD context. The traditional methods should be utilised for specific and well-aimed purposes. For example, the GQM method can be utilised while identifying new GSD specific metrics.

In GSD, development processes are dynamic and thus results of measurements and their interpretation vary. In this paper, GSD metrics used in the companies were focused on 'early warning' signals for the project and management. In a changing environment it's also an important aspect that the

measurement data is easy to collect and that the metrics can be quickly calculated at regular intervals. Ease of use and speed are also central factors from metrics interpretation viewpoint. This also emphasises the importance of metrics visualisation. Interestingly, GSD literature has rarely focused on metrics and measurements or given experimental examples of successfully used metrics during GSD development.

C. Balancing Measurements

A Balanced Scorecard (BSC) is widely used for monitoring performance of an organisation towards strategic goals. The original BSC approach covers a small number of performance metrics from four perspectives, called as Kaplan & Norton perspectives: Financial, Customer, Internal Processes, Learning & Growth [46]. The BSC framework added strategic non-financial performance measures to traditional financial metrics to give managers and executives a more 'balanced' view of organizational performance. However, many early BSCs failed, because clear information and knowledge about the selection of measures and targets were not available. For example, organisations had attempted to use Kaplan & Norton perspectives without thinking about whether they were suitable in their situation. After that many improvements and enhancements have been completed on BSC approach. Since 2000, it has been described as a "Third Generation" of Balanced Scorecard designs. The BSC has evolved to be a strategic management tool that involves a wide range of managers in the strategic management process, provides boundaries of control, but is not prescriptive or constrictive and more importantly, removes the separation between formulation and implementation of strategy [47]. The BSC suggests that organisation should be viewed from four perspectives (Learning & Growth perspective, Business process perspective, Customer perspective, and Financial perspective) and metrics should be developed, data collected and analysed in relation to these perspectives.

Even if BSC are generally intended to deal with strategic issues, in this paper, the balancing of various perspectives of BSC has been emphasised. In fact, it has been proved that Practical Software Measurement and the Balanced Scorecard are both compatible and complementary [48]. In GSD context, decisions or actions taken based on the analysis of metrics and measurements collected from different development parties or stakeholders need to take specific the GSD factors into account as well.

D. Measurement Challenges in GSD

Even in the daily software development work, the measurements are still seen as unfamiliar or an extra burden for projects. For example, project managers feel it is time consuming to collect metrics for the organization (business-goal-related metrics), yet they need to have metrics that are relevant to the project. Furthermore, in many cases, not enough time is budgeted for measurements, and this is why it is very difficult to obtain approval from stakeholders for this kind of work [5].

Globally distributed development generates new challenges and difficulties for the measurements. For

example, the gathering of the measurements data can be problematic because of different development tools which have different versions, work practices with related concepts can vary by project stakeholders or reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations. In addition, distributed projects are often so unique (e.g., product domain and hardware-software balance vary, or different subcontractors are used in different phases of the project) that their comparison is impossible. Thus, the interpretation of measurements data is more complicated in GSD than one-site projects. This is why it is recommended to select a moderate amount of metrics. In this paper, we will present a set of metrics as well as examples of their visualisation possibilities to support decision making in GSD. Also industrial experiences about the metrics will be discussed.

The common metrics (effort, size, schedule, etc.) are also applicable for GSD projects. However, special attention may be needed in training the metrics collection, to ensure a common understanding of them (e.g., used classifications). In addition, as measurements also tend to guide people's behaviour, it is important to ensure that all are aware of the purpose of the metrics (i.e., not to measure individual performance), specifically in projects distributed over different cultures. In GSD content the automation of measurements is highly recommended to avoid misunderstanding - even if it is not easy to implement. The focus is to generate real-time information shown in a format that is easy and quickly interpreted. This means that great attention should be paid to metrics visualisation.

V. GENERIC MEASUREMENTS AND METRICS IN GSD

In this section, the metric set used in the companies is introduced. In addition, several visualised examples of proposed metrics are given and discussed. The metric set and their visualisation examples have been produced during the ITEA PRISMA (2008-2011) project [49]. The main goal of the PRISMA project was to boost productivity of collaborative systems development. One of the project's results was the Prisma Workbench (PSW), a tool integration framework [50]. PSW provides several real-time views into data that has been collected from various data sources even from separate stakeholders' databases. The PSW enabled the visualisation of metrics in GSD and collection of the experiences of their use. The work was done in close cooperation with industrial partners and experimental views were generated based on their needs or challenges. The original metrics were the same that the industrial partners had successfully used in their globally distributed projects, published in [1]. During the PRISMA project, the development of the PSW tool enabled further development of the proposed metrics set and their visualisation in cooperation with the industrial partners. The industrial partners had identified metrics, and defined their collection and visualisation. They had also tried the metrics in few projects to collect experiences. These experiences were then shared among the industrial partners of the project. The researchers analysed the measurements and experiences to find commonalities from these measurement practices. Results of

this analysis was discussed in workshops with the companies, and updated based on the comments. This paper presents the results of this work. In following sub-sections, the developed example views are shown and discussed. Industrial experiences, opinions and ideas for improvement are also presented. The industrial experiences were gathered during the industrial cases by interviewing companies' personnel who had developed the metrics and measurement programs.

A. Rational Unified Process (RUP) Approach

Each phase in the lifecycle of a development project affects the interpretation of the metrics. Thus, in this paper, proposed metrics and visualisation examples are introduced by using commonly known approach of software development called Rational Unified Process (RUP). Also the processes used in the companies were similar to the RUP phasing, so it was chosen as a presentation framework for this paper. RUP is a process that provides a disciplined approach to assigning tasks and responsibilities within a development organisation. Its goal is to ensure the production of high quality software that meets the needs of its end-users within a predictable schedule and budget [16][51].

The software lifecycle is divided into cycles, each cycle working on a new generation of the product. RUP divides one development cycle in four consecutive phases [51]: (1) inception phase, (2) elaboration phase, (3) construction phase and (4) transition phase. There can be one or more iterations within each phase during the software generation. The phases and iterations of RUP approach are illustrated in following Figure 1.

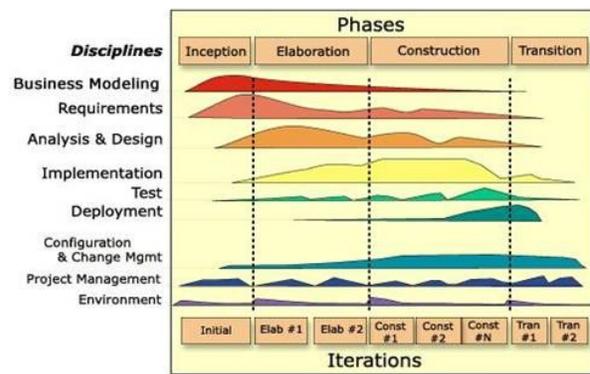


Figure 1. Phases and Iterations of RUP Approach [51]

From a technical perspective, the software development is seen as a succession of iterations, through which the software under development evolves incrementally [16]. From measurement perspective this means that some metrics can be focused on during one or two phases of the development cycle, and some can be continuous metrics that can be measured in all phases, and can be analysed in each iteration.

In this paper, the metrics are introduced according to the RUP phases. Each metric is presented in the phase where the

metric can be utilised in the first time or where the metric is seen to be the most relevant to measure, even if some metrics are relevant in several phases. In fact, many of the introduced metrics can be used also in the following product development phases. For each metric - a name, a notation and a detailed definition is introduced. The main goal is to offer a useful, yet reasonable amount of metrics, for supporting the on-time monitoring of the GSD projects. The indicators are supposed to be leading indicators rather than lagging indicators. For example, planned/actual schedule measurements should be implemented as milestone trend analysis which measures the slip in the first milestone and predicts the consequences for the other milestones and project end.

B. Metrics and their Visualisation for Inception Phase

During the inception phase, the project scope has to be defined and the business case has to be established. The business case includes success criteria, risk assessment, and estimate of the resources needed, and a phase plan showing dates of major milestones. Inception is the smallest phase in the project, and ideally it should be quite short. Example outcomes of the inception phase are a general vision document of the project's core requirements, main constraints, an initial use case model (10% -20% complete), and a project plan, showing phases and iterations [52]. Proposed metrics to be taken into consideration in this phase are introduced in Table I.

TABLE I. METRICS FOR THE INCEPTION PHASE

Metric	Notation	Definition
Planned Schedule	D_{PLANNED}	The planned Date of delivery (usually the completion of an iteration, a release or a phase)
Planned Personnel	$\# FT_{\text{PLANNED}}$	The planned number of Full Time persons in the project at any given time
Planned Effort	E_{PLANNED}	The planned Effort for project tasks (/requirements) at any given time
Proposed Requirements	$\# \text{Reqs}$	The number of proposed requirements.

The metrics Planned Schedule and Planned Personnel /Effort are mostly needed for comparison with actual schedule, personnel and effort, in order to identify lack of available resources as well as delays in schedule quickly. The amount of Proposed Requirements tells about the progress of the product definition.

Figure 2 shows how some of the proposed metrics can be utilised during product development for visualising the progress of project. The metric of progress status combines effort and schedule metrics in a visualised way. The first and top line (blue) in the Figure 2 is a cumulative planned effort over time calculated from project tasks. The next line, the red line describes the cumulative updated planned effort and accordingly, the green line describes the cumulative actual used effort over time summarised from project tasks. The bottom and last line in lilac shows the earned value that indicates the cumulative effort of completed tasks (/workproducts).

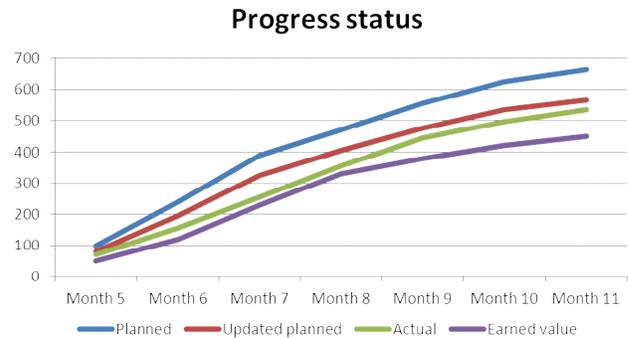


Figure 2. Visualised Metric: Progress Status

The graph visualises the project progress and easily gives several kinds of information as well as proactive insights, such as, is the project resourcing in place, and is the project completing work as planned. In the shown graph, it is a good signal that cumulative planned effort (blue line) is continuously above the cumulative updated planned effort (red line); it means the project is running on schedule. Another good signal is if actual used effort (green line) and earned value (lilac line) is relatively close to each others; it means that the results (~completed tasks) have been achieved with the used effort. The status in the Month 11 indicates that there are still several open tasks that are not completed even if actual used effort (green line) seems to draw closer to the cumulative updated planned effort (red line); this indicates a potential threat. Depending on project's phase (for example, in the middle phase or at the ending phase) corrective actions would be needed. The actions are not needed if the project is at the ending phase because the cumulative planned effort (blue line) is still clearly the upmost line.

Industrial comments

In the Philips company example, the Progress status metric has proven to give a timely insight in the actual consumption of effort compared to planned effort in large first-of-a-kind Consumer Electronics projects. The representation over time enables the ability to analyse trends, and take actions pro-actively. Moreover, the use of earned value gives insight in the effectiveness of the effort spent answering the question: "Does the effort spent contribute to realizing the agreed results?"

In the Symbio company example, indicators of earned value and tracking of unplanned work were seen as especially important from a management perspective. Unplanned work may yield a strong indication of a variety of causes early in the project, such as technical infeasibility or a lack of shared vision between project stakeholders. Accordingly, they identified that from a budget perspective, justifying workshops early in the project to shape a shared vision and collaborate on scoping project goals is often difficult to qualify for many stakeholders. It is a typical case that only when problems manifest, or a sharp trend in unplanned work is experienced will stakeholders react. Usually, remedying the problem requires unplanned trips to

put people together into the same room to hammer out solutions that essentially consume budget.

C. Metrics and their Visualisation for Elaboration Phase

During the elaboration phase a majority of the system requirements are expected to be captured. The purpose of the phase is to analyse the problem domain, establish a sound architectural foundation, develop the project plan, and eliminate the highest risk elements of the project. The final elaboration phase deliverable is also a plan (including cost and schedule estimates) for the construction phase. Example outcomes of the elaboration phase are; a use case model (at least 80% complete), a software architecture description, supplementary requirements capturing the non-functional requirements and any requirements that are not associated with a specific use case, a revised risk list and a revised business case, and a development plan for the overall project. Proposed metrics to be taken into consideration in this phase are introduced in Table II.

TABLE II. METRICS FOR THE ELABORATION PHASE

Metric	Notation	Definition
<u>Schedule:</u> Planned /Actual Schedule	D_{PLANNED} D_{ACTUAL}	The planned/actual Date of delivery (usually the completion of an iteration, a release or a phase)
<u>Staff:</u> Planned /Actual Personnel Planned /Actual Effort	#FT_{PLANNED} #FT_{ACTUAL} E_{PLANNED} E_{ACTUAL}	The planned/actual number of Full Time persons in the project at any given time. The planned/actual Effort for project tasks (/requirements) at any given time.
<u>Requirements</u> -Drafted -Proposed -Approved -Not implemented	#Reqs_{DRAFTED} #Reqs_{PROPOSED} #Reqs_{APPROVED} #Reqs_{NOT_IMPL}	The number (#) of - drafted requirements - proposed requirements - reqs approved by customer - not implemented reqs
<u>Tests</u> -Planned	#Tests_{PLANNED}	The number (#) of - planned tests
<u>Documents:</u> -Planned -Proposed -Accepted	#Docs_{PLANNED} #Docs_{PROPOSED} #Docs_{ACCEPTED}	The number (#) of planned /proposed /accepted documents to be reviewed during the project.

The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. The Staffing metric may explain deviations in the expected progress vs. the actual progress, both from a technical as well as from a schedule viewpoint. Note that those metrics that are more relevant to measure by iterations (effort and size) are introduced later (in Section E).

Figure 3 shows how some of the proposed metrics can be visualised in order to describe the project’s status. The metric of requirements status combines the amount of planned effort with status of requirements’ implementation over a time in the same graph. The bars summarise the amount of planned effort for the month. Each bar is composed from four different data relating to identified requirements as follows. The first block (green) describes a sum of planned efforts for all implemented requirements. The second block (grey) describes a sum of planned efforts for approved but not implemented requirements. The third block (blue)

describes a sum of planned efforts for proposed requirements and the last block (orange) shows a sum of planned efforts for drafted requirements.

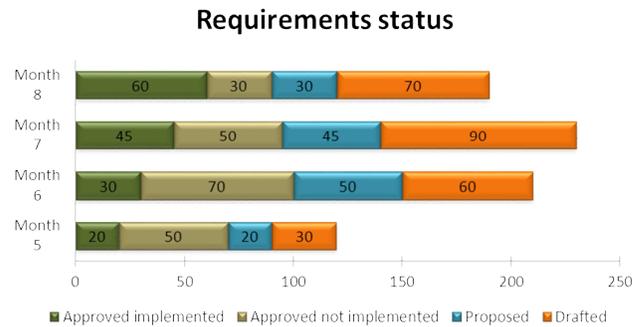


Figure 3. Visualised Metrics: Requirements Status

It is important to note, that the planned effort is used constantly, even for implemented requirements. This is due to keeping the baseline in order to enable comparing project situation over time, i.e., to be able to see the project trend with respect to planned work. The planned effort may be updated for the requirements during the project, if a new baseline is created. This information is then used together with the actuals, to see how well the planning has succeeded to help learning to estimate better.

The visualised metric “Requirements status” indicates several status information but also trend lines relating to requirements implementation, and is focused on showing the uncertainly of the project, for example how much more work maybe dedicated to be implemented in the project. In the example graph, a good signal is that the sum of planned efforts for implemented requirements seems to increase over time while the sum of planned efforts for approved, but not implemented requirements, seems to reduce. However, the sums of planned efforts for proposed and drafted requirements are still quite large in the Month 8, especially, while comparing them to the sums of planned efforts for approved requirements. This indicates that the project is in the beginning phase rather than in the ending phase. However, the interpretation needs other metrics information, such as “Progress status” or “Testing status” to make any decisions.

Industrial comments

In the Philips company example, the current projects lack insight into the satisfaction of requirements. This lack of insight concerns both the actual status of implementation of the requirements, as well as the expectation: “Up to what level the project will be able to satisfy its requirements, and if not, what are measures to accomplish that?” The (leading) indicator as proposed in this document seems to be a good answer to this problem. The metric has been introduced in a few (one-roof) projects yet and initial results seem promising. However, no data with experiences on a metric like this have been collected yet.

According to Symbio’s practice, when looking to exit an elaboration phase, product owners should pay special

attention to the coverage of requirements affecting architecture to ensure the construction phases run more to plan as the team sizes may scale and involve more sites. Whilst iterative development can be seen as promoting elaboration of requirements later in the lifecycle, core functions that separate the project output from competition should be conceptualized and approved for implementation. Project managers may consider implementation of these differentiating use cases to be made geographically or temporally close to the project owner. Non-approved requirements should be managed accordingly and not planned for implementation off-site until they are suitably elaborated and accepted into the development roadmap. Misunderstanding of the requirements needs to be minimized if the team size and development sites scale during construction phases otherwise projected cost savings from multi-site development can be quickly eliminated.

D. Metrics and their Visualisation for Construction Phase

Construction is the largest phase in the project. During the phase, all remaining components and application features are developed and integrated into the product, and all features are thoroughly tested. System features are implemented in a series of short, time boxed iterations. Each iteration results in an executable release of the software. Example outcomes of the phase consist of a software product integrated on the adequate platforms, user manuals, and a description of the current release. Proposed metrics to be taken into consideration in this phase are introduced in Table III.

TABLE III. METRICS FOR THE CONSTRUCTION PHASE

Metric	Notation	Definition
Planned /Actual Schedule Planned /Actual Personnel	$D_{PLANNED}$ D_{ACTUAL} $\#FT_{PLANNED}$ $\#FT_{ACTUAL}$	Defined in the elaboration phase.
Requirements: -Proposed -Approved -Not implemented -Started -Completed	$\#Reqs_{PROPOSED}$ $\#Reqs_{APPROVED}$ $\#Reqs_{NOT_IMPL}$ $\#Reqs_{STARTED}$ $\#Reqs_{COMPLETED}$	The number (#) of - proposed requirements - reqs approved by customer - not implemented reqs - reqs started to implement - reqs completed
Change Requests: -New CR -Accepted -Implemented	$\#CRs_{NEW}$ $\#CRs_{ACCEPTED}$ $\#CRs_{IMPL}$	The number (#) of - identified new CR or enhancement - CRs accepted for implementation - CRs implemented
Tests: -Planned -Passed -Failed -Not tested	$\#Tests_{PLANNED}$ $\#Tests_{PASSED}$ $\#Tests_{FAILED}$ $\#Tests_{NOT_TESTED}$	The number (#) of - planned tests - passed tests - failed tests - not started to test
Defects -by Priority: e.g., Showstopper, Medium, Low	$\#Dfs_{PRIORITY}$	The number (#) of - defects by Priority during the time period
Documents: -Planned -Proposed -Accepted	$\#Docs_{PLANNED}$ $\#Docs_{PROPOSED}$ $\#Docs_{ACCEPTED}$	Defined in the elaboration phase.

Note that those metrics that are continuously measured are introduced later (in Section E).The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. Metrics related to changes indicate both the stability of the project technical content, and can explain schedule delays, and unexpected technical progress. Defect metrics describe both the progress of testing as well as the maturity of the product.

In the construction phase, all components and features are developed and integrated into the product. In addition, they are also thoroughly tested, so there are many simultaneous actions that can be implemented by multiple partners or/and in different locations in GSD. This is why the metrics interpretation needs to be done very carefully by utilising indicators from different data sources and from different partners. In this subsection two metrics: “Budget status” and “Testing status” are introduced with discussion about indicators and proactive signals that they provide.

The visualisation of Budget status combines cost, requirements and defects metrics in the same graph shown in Figure 4.

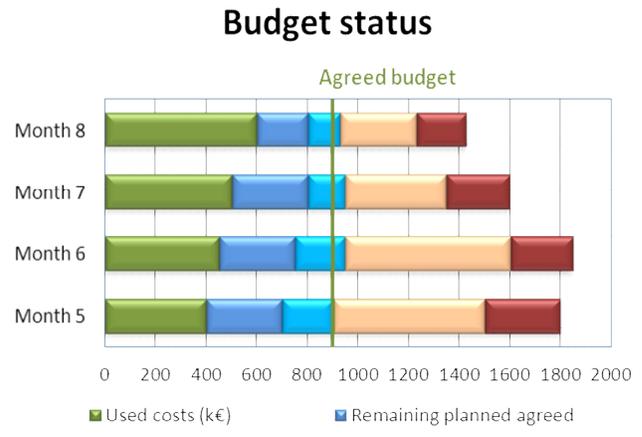


Figure 4. Visualised Metrics: Budget Status

The Budget status graph shows actual costs of the project in portion with the agreed budget over a time period. The metric also gives several indicators of estimated prospective costs in each month. The bars summarise amount of costs for the month, and each bar is composed from five different cost-related data. The first block (green) describes actual cumulative costs of the project. The agreed budget for the project is shown clearly as a green line in the middle of the graph. The second block (blue) describes remaining planned cost based on effort estimated for requirements that have been accepted for implementation but not yet implemented. The third block (light blue), in the middle of the bar, indicates proposed cost that can be seen very likely costs for the project. These costs are based on effort estimated for the proposed requirements that are estimated likely to be implemented, for example, a customer will want them. The fourth block (orange) describes proposed but vague costs for

the project. These costs are based on effort estimated for the proposed requirements which the likeliness for implementation is not known. Instead, the fifth block (red) indicates very potential costs for the project, so-called “Known defects” costs. The costs are based on effort estimated to be needed to fix the known critical, major or average defects. In the example graph, the Budget status metric in Figure 4, the project’s costs will overrun the agreed budget.

Industrial comments

At Philips, the current applied budget metrics generally give a clear understanding in the actual budget consumption, but are poor in predicting budget consumption for the remainder of the project. The metric suggested allows for trend analysis and by that extrapolation to the future, resulting in better prediction of the budget consumption for the remainder of the project. This will improve the projects’ and the management’s insight into the project and enable them to take required measures in a timely fashion, as appropriate. The metric has not yet been applied in our projects.

In Symbio, managers will often track cost against budget throughout construction for project sponsors, but earned value becomes increasingly more important in the latter stages of the lifecycle. Earned value can be tracked with relative ease if defined requirements are quantified for business importance. Product backlogs imply the importance by a requirement’s position in the backlog; however some backlogs may include other items than requirements such as operational tasks for deployment and so on. To compensate all project requirements (both functional and non-functional) can be attributed with a business value, its value based in comparison to the cumulative value of all project requirements. When a requirement is delivered its value is added to cumulative total to provide an earned value delivered by the project. This approach is ideal if the backlog of the product development stabilizes throughout construction. However significant changes in the business value of requirements will weaken the importance of tracking this metric over time. Also this metric requires the project team and stakeholders to agree upon a “definition of done” which can be very difficult, and even more so if the accepting and implementing parties are different entities or located in different sites.

The metric of Testing status combines effort, requirements and test metrics in a same graph. The Testing status metric visualises the progress of testing phase by collecting data from various phases. The bars in the graph summarise efforts relating to tests in each month. Each bar is composed from four different sums of efforts. The first block (green) describes a sum of efforts for tested requirements. The second block (blue) describes a sum of efforts for requirements for which test case is available, and accordingly, the third block (purple) describes a sum of efforts for requirements for which test cases are not available. The last, the fourth block (red) is a very proactive indicator describing a sum of effort estimated for uncertain requirements. Figure 5 shows the visualisation of Testing status metric.

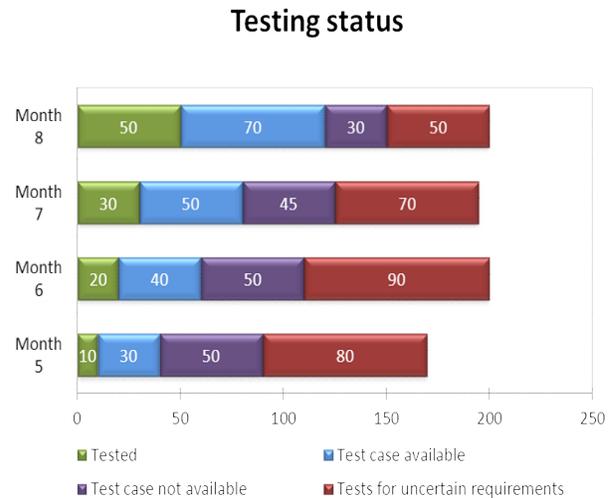


Figure 5. Visualised Metrics: Testing Status

Even if “Testing status” shows easily how ‘mature’ the testing phase is the metric requires other metrics – such as the before introduced metrics: Budget status, Progress status and Requirements status – make conclusions based on the data.

Industrial comments

According to Philips, one of the most important indicators of a development project is insight in what will be the status of the product at the delivery time - what will the product actually contain and what is the quality of those contents? This metric is an effective means to get early insight in the status of the product by the end of the project. Moreover, the test status trend analysis helps to initiate timely measures to work towards an agreed project result. The metric has been applied in a single project at Philips and results were promising - it really improved the insight of project, management and customer in the status of the product-under-construction and better understanding of what could be expected by the end of the project.

According to Symbio, earned value is especially invaluable in the close down phases of a project. Projects may deteriorate into loss making, unplanned iteration as stakeholders become overly conscious on metrics of requirements coverage. This situation is can be further exacerbated if the value of requirements is not continually reviewed and communicated to all stakeholders throughout the project.

E. Metrics for Transition Phase

The final project phase of the RUP approach is transition. The purpose of the phase is to transfer a software product to a user community. Feedback received from initial release(s) may result in further refinements to be incorporated over the course of several transition phase iterations. The phase also includes system conversions, installation, technical support, user training and maintenance. From measurements viewpoint the metrics identified in the phases relating to schedule, effort, tests, defects, change requests and costs are still relevant in the transition phase. In addition, customer

satisfaction is generally gathered in the transition phase, and post-mortem analysis carried out.

F. Metrics for Iterations

Each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release. Each release is accompanied by supporting artifacts: release description, user's documentation, plans, etc. Although most iterations will include work in most of the process disciplines (requirements, design, implementation, testing) the relative effort and emphasis will change over the course of the project. Proposed metrics for each iteration to be taken into consideration, are introduced in Table IV.

TABLE IV. METRICS FOR ITERATIONS

Metric	Notation	Definition
<u>Effort:</u> -Planned Effort -Actual Effort	$E_{PLANNED}$ E_{ACTUAL}	The planned/actual effort required of any given iteration of the project.
<u>Size:</u> -Planned size -Actual size	$SIZE_{PLANNED}$ $SIZE_{ACTUAL}$	The planned /actual size of each iteration can be measured as SLOC (Source Lines of Code), Points or any other commonly accepted way.
<u>Cost:</u> -Budgeted -Expenditure	$COST_{BUDGET}$ $COST_{ACTUAL}$	The budgeted cost /actual expenditure for any given iteration.
<u>Velocity:</u> -planned /actual story points	$\#PTS_{PLAN}$ $\#PTS_{ACT}$	How many story points are planned to be /actually implemented of any given iteration of the project.
<u>Productivity:</u>	$E_{ACTUAL} / \#PTS_{ACT}$	Use effort per actually implemented story points for each sprint /iteration

All of these metrics provide indications of the project progress and reasons for deviations should be analysed. These metrics should be analysed together with other metrics results (presented in Tables I-III) in order to gain a comprehensive picture of the status.

VI. SPECIFIC MEASUREMENTS AND METRICS IN GSD

Section V discussed metrics, which are not specific for GSD, but they provide valuable information to follow a GSD project progress. So, in GSD, metrics can be similar or same as in single-site development. However, in order to prevent potential problems during distributed projects some specific GSD metrics could be added to be used together with the metrics presented in Section V. These metrics should be focused on the specific challenges in GSD that were presented in general level in Section III and they would help to quickly detect the GSD related source of problems that are identified in the metrics presented in Section V.

Measuring the generic GSD challenges (Section III) is difficult, and in fact, measuring the challenges does not provide clear value from project monitoring viewpoint. It is more beneficial to follow and detect the symptoms that indicate problems in the GSD practice. Example problems [14] caused by lack of communication, coordination

breakdown, and different backgrounds include, for example, ineffective use of resources as competences are not known from other sites, obstacles in resolving seemingly small problems and faulty work products due to a lack of competence or background information. These causes can also lead to a lack of transparency in the other parties' work, misunderstood assignments and, thus, faulty deliveries from parties, delays caused by waiting for the other parties' input and duplicate work or uncovered areas. Further problems that can be caused by these issues include differences in tool use or practices in storing information, misplaced restrictions on the access to data and unsuitable infrastructure for the distributed setting.

Example problems [14] caused by lack of teamness and lack of trust include hiding problems and unwillingness to ask for clarification from others, expending a lot of effort in trying to find that the cause of problems (defects) has occurred in the other parties' workplace, an unwillingness to help others and an unwillingness to share information and work products until specifically requested to do so. These causes may also appear as difficulties in agreeing about the practices to be used and then not following the process and practices as agreed, for example. Further problems caused by these issues include the use of other tools than those agreed to for the project and plentiful technical issues that hinder communication and use of the tools, as agreed.

The following problems are among the most common ones in companies GSD practice (based on 54 industrial cases during several research projects):

1. unclear responsibilities and escalation channels,
2. unavailability of information timely for all who need it,
3. unclear information and misunderstandings (for example of requirements and task assignments),
4. problem hiding,
5. non-communicated and unexpected changes,
6. lack of visibility and transparency of all sites work and progress,
7. faulty and/or delayed (internal) deliveries, and
8. sub optimal use of resources.

Next we discuss potential measurements to indicate as early as possible if these problems are present. These proposals have not been applied in practice, yet. Instead, their implementations and possible selections were discussed with industrial partners.

Relating to problems 1-3, a measurement could be a short questionnaire asking the project members if they know their responsibilities and when and to whom to escalate problems, and is the required information available and clear. In addition, from GSD viewpoint, a potential measurement could be related to time spent idling (a team member is waiting because of wrong, incorrect or missing information or input from other members) or percentage of unplanned work (a team member is working with unplanned or duplicated tasks).

For problems 4 – 6, a measure is amount and type of communication over sites. For example, communications activeness could be monitored via metrics like amount of status reports, meeting memos, chats, calls between

locations, etc. Communication activeness is especially important between distributed teams where their development tasks are highly coupled or dependent on deliveries and results of each other. For example, silence or communication only via documents (official reports) can be an indicator of problem, whereas active informal communication over sites indicates active discussion of work at hand. In the worst case in GSD, lack of face-to-face communication can lead to “reportmania” where communication is handled only through large amount of documents. Long textual descriptions can be easily omitted or alternatively misunderstood because of high amount of effort and time required for adopting the content.

For problem 7, metrics related to defects and schedule are relevant and for problem 8, a potential measurement is time spent idling and the time blocked because of the impediments elsewhere in the team as these affect productivity and highlight when a team is not performing. Also, communication related metrics are valuable for these problems.

The metrics relating to team trust, project commitment and team identifications describes team dynamics that can provide lot of explaining information for the problems in GSD project. Some indicators, such as how many people have left from the project, refer the individual satisfaction as well as project commitment. Because software development is fundamentally team oriented action [53], metrics relating to team dynamics and teamwork quality is highly recommended to monitor in GSD. Potential metrics are related to communication, tasks coordination, balance of member contributions, mutual support, effort and cohesion as introduced by Hoegl and Gemuenden [54]. Examples of questions are as follows:

- Communication: Is there sufficient frequent, informal, direct, and open communication?
- Coordination: Are individual efforts well-structured and synchronised within the team?
- Balance of member contributions: Are all team members able to bring in their expertise to its full potential?
- Mutual support: Do team members help and support each other in carrying out their tasks?
- Effort: Do team members exert all efforts to team tasks?
- Cohesion: Are team members motivated to maintain the team? Is there team spirit?

These questions can be used to measure team dynamics and team work quality during a GSD project.

VII. DISCUSSION

A. GSD Metrics

As discussed, little focus has been paid on GSD metrics in the literature. In fact, the research has been focused on clarifying differences between collocated and distributed projects and also, identifying variables that differ the most. Although this kind of approach is important for gaining knowledge about the issues that need to be monitored in GSD, a specific focus on the metrics and their collection and analysis is also needed. For example, project performance is

even more complicated and multi-level concept to measure in GSD than in single-site. It concerns team members’ individual performance, teamwork performance and tasks performance as well as management performance. Bourgault et al. [19] pointed out that distributed projects’ performance metrics and measurement needs more attention so that well designed management information systems could be developed in order to create effective monitoring systems for distributed projects. This kind of development was seen as necessary to provide decision makers with dynamic, user-friendly information system that would support management activities, not only for project managers, but also for top managers. However, the issue of performance metrics in the context of distributed projects needs to be investigated in more detail. Furthermore, a dispersion of work has significant effects on productivity and, indirectly, on the quality of the software. However, it is currently difficult to specify metrics, measurement processes and activities that best suit different companies and specific GSD circumstances. We have presented a first step towards taking into account the specific aspects of GSD in measurement programs, but more work is needed. For example, specific GSD metrics are currently collected and processed manually, thus requiring extra and error prone effort. In the world of the hectic and dynamic GSD practice, the metrics collection and visualisation should also be automated to be valuable in large-scale use. The automation is an important issue for further research.

B. Industrial Viewpoint

The metrics presented in Section V were common for both of the companies. Although the metrics were chosen independently by both companies, the reasoning behind choosing these metrics was similar. An important reason was to come from a re-active into a pro-active mode, for example to introduce ‘early warning’ signals for the project and management. Specifically these metrics have been chosen as they indicate a well-rounded view of status in the various engineering disciplines and highlight potential issues in the project. This creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. This is especially important in GSD, where information of project status is not readily available but needs special effort, distributed over sites and companies. Accordingly, the metrics set can be seen as a ‘balanced score card’, on which management can take the right measures, balancing insights from time, effort (e.g., staffing), cost, functionality (requirements) and quality (tests) perspective.

An important aspect was also that the metrics are easy to capture and that they can be captured from the used tools “for free”, or can be quickly calculated at regular intervals. Costs and budgets are good examples of metrics that can be easily captured from the tools. This is also important from GSD viewpoint, as automated capturing reduces the chance of variations caused by differences in recording the metrics data in different sites. Neither of the companies use metrics based on lines-of-code as they did not find it to be a reliable indicator of progress, size or quality of design.

It can be seen that the metrics are quite similar as in single-site development. However, the metrics may be analysed separately for each site, and comparisons between sites can thus be made in order to identify potential problems early. On the other hand, it is important to recognise that some metrics correlate with each other, for example, metrics relating to tests correlate with metrics about requirements, and that needs to take consideration while analysing. In general, the interpretation of project's comprehensive status needs various metrics information – like Requirements status, Progress status, Testing status and Budget status – for making conclusions based on the data. In addition, while interpreting or making decisions based on the measurement results the distributed development implications need to be taken into account. Distributed development requires 'super-balancing' - how to come to the right corrective action if for instance, on the one side, the % of not accepted requirements is high, and on the other side, the # of passed tests is lagging behind. Distributed development may also affect the actual results of the measurements. For example, relating to subjective metrics, such as effort estimation, differences between backgrounds of the people (cultural or work experience) in different sites may affect the result.

The companies also use the measurement results to gain insight into why a measure varies between similar single-site and multi-site projects in order to try to reduce potential variances. This also partially explains the use of the same metrics as single-site development. This was experienced by the representative of Symbio: *"These points presented should be by now well known. From an economic perspective these points must be considered when evaluating and comparing costs of different project models of delivery."*, and *"Benchmarking and tracking of historical data across the entire project portfolio is still only an initial step to shape more informed cost estimations when composing project teams with distributed elements. Continuous effort is required not only in definition and capture of metrics but also in the effects on working practices in general."*

Furthermore, the challenges in communication and dynamics of distributed teams mean that working practices need to be addressed continuously as impressed by Symbio representative: *"Often a practical solution to working procedures can result in compensation for potential lost productivity. For example a testing team in China lags their working week by one day (Tuesday to Saturday) in order to test the results from an implementation team in Finland (working Monday to Friday). In this example the Finland team agrees to ensure continuous integration in order to not block the testing team. If these two practices have a positive effect on productivity when compared against similar project models, future cost estimations should then be benchmarked on the new working practices."* However, in addition to metrics results, paying close attention and acting on feedback is as important, if not more important than drawing strong conclusions from metrics alone.

Currently, both companies are in process of revamping their metric usage, but feel confident that the metrics introduced in this paper are the right ones. This was pointed out by Philips by the following: *"Applying the metrics*

suggested in this document to the parties involved in the GSD project already gives better insight in the relative performances of the groups, and enables to take measures over time (e.g., systematically improve a party's performance, or replace it). We have applied detailed effort consumption metrics to our single-roof and multi-side development projects. Those metrics learned that staff of multi-side projects spend significantly more time on things they call 'communication' or 'overhead' (up to 50%!). Our understanding of the matter is that no new metric needs to be 'invented' for that: standard effort distribution metrics would do. The main challenge is to have it introduced in a systematic way, with the same understanding and interpretation of the metrics by the parties involved. Especially the first element is often a challenge: third parties are often reluctant to provide this level of transparency of their performance."

Both companies are careful in introducing new metrics, as it is well known that too many metrics lead to overkill and rejection by the organization, and do not provide the right insights and indication for control measures. Easy implementation and by that, easy acceptance is the most crucial thing to get these metrics as established practice within the company. However, the few specific GSD metrics presented in Section VI are intended to be used together as the proposed metrics set. These additional metrics should be focused on measuring the project performance, especially task and team performance in GSD.

VIII. CONCLUSION

The management of the more and more common distributed product development project has proven to be more challenging and complicated than traditional one-site development. Metrics are seen as important activities for successful product development as they provide the means to effectively monitor the project progress. However, defining useful, yet reasonable amount of metrics is challenging, and there is little guidance available for a company to define metrics for its distributed projects.

Globally distributed development generates new challenges and difficulties for the measurements. For example, the gathering of the measurements data can be problematic because of different development tools and their versions, work practices with related concepts can vary by project stakeholders or reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations. Furthermore, interpretation and decision-making based on the measurement results require that the distributed development implications are taken carefully into consideration.

This paper focused on describing a set of metrics that is successfully used in industrial practice in GSD and given examples of their visualisation with industrial experiences of their use. These metrics, are aimed especially to provide the means to proactively react to potential issues in the project, and are meant to be used as a whole, not interpreted as single information of project status. The basic GSD circumstances with challenges are discussed from viewpoints of metrics

and measurements in order to create awareness and knowledge of potential GSD specific metrics.

The metrics presented in the paper were common for both of the companies. Based on experiences, the reasoning for selecting these metrics was similar: they are easy to capture and can be quickly calculated and analysed at regular intervals. Also, one of the most important reasons was that these metrics were aimed especially to provide the means to proactively react to potential issues in the project. The balancing insights from time, effort, cost, functionality and quality was also seen as very important aspect.

ACKNOWLEDGMENT

This paper was written within the PRISMA project that is an ITEA 2 project, number 07024 [49]. The authors would like to thank the support of ITEA [55] and Tekes (the Finnish Funding Agency for Technology and Innovation) [56].

REFERENCES

- [1] M. Tihinen, P. Parviainen, R. Kommeren and J. Rotherham, "Metrics in distributed product development," In Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA'11), Barcelona, Spain, 2011, pp. 275-280.
- [2] R. Van Solingen and E. Berghout, *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill, 1999.
- [3] V. R. Basili, *Software modeling and measurement: The Goal /Question/Metric paradigm*. Computer Science Technical Report CS-TR-2956, UNIMACS-TR-92-96, University of Maryland at College Park, Sep. 1992, pp. 1-24.
- [4] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co. Boston, MA, USA, 1998.
- [5] M. Umarji and F. Shull, "Measuring developers: Aligning perspectives and other best practices," *IEEE Software*, vol. 26, (6), 2009, pp. 92-94.
- [6] J. Hyysalo, P. Parviainen and M. Tihinen, "Collaborative embedded systems development: Survey of state of the practice," In Proceedings of 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 2006), IEEE, 2006, pp. 1-9.
- [7] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," In Proceedings of Future of Software Engineering FOSE '07, IEEE Computer Society, 2007, pp. 188-198.
- [8] J. D. Herbsleb, A. Mockus, T. A. Finholt and R. E. Grinter, "Distance, dependencies, and delay in a global collaboration," In Proceedings of the ACM Conference on Computer Supported Cooperative Work, ACM, 2000, pp. 319-328.
- [9] M. Jiménez, M. Piattini and A. Vizcaíno, "Challenges and improvements in distributed software development: A systematic review," *Advances in Software Engineering*, vol. Jan-2009, (No. 3), 2009, pp. 1-16.
- [10] S. Komi-Sirviö and M. Tihinen, "Lessons learned by participants of distributed software development," *Knowledge and Process Management*, vol. 12, (2), 2005, pp. 108-122.
- [11] M. Tihinen, P. Parviainen, T. Suomalainen, K. Karhu and M. Mannevaara, "ABB experiences of boosting controlling and monitoring activities in collaborative production," In Proceedings of the 6th IEEE International Conference on Global Software Engineering (ICGSE'11) Helsinki, Finland, 2011, pp. 1-5.
- [12] F. Q. B. da Silva, C. Costa, A. C. C. França and R. Prikladinicki, "Challenges and solutions in distributed software development project management: A systematic literature review," In Proceedings of International Conference on Global Software Engineering (ICGSE2010), IEEE, 2010, pp. 87-96.
- [13] S. Komi-Sirviö and M. Tihinen, "Great challenges and opportunities of distributed software development - an industrial survey," In Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering (SEKE2003), San Francisco, USA, 2003, pp. 489-496.
- [14] P. Parviainen, "Global software engineering. challenges and solutions framework," *Doctoral Dissertation*, VTT Science 6, Finland, 2012, pp. 106 p. + app. 150 p.
- [15] Prisma-wiki, SameRoomSpirit wiki homepage. URL: http://www.sameroomspirit.org/index.php/Main_Page (Accessed 19.12.2012).
- [16] P. Kruchten, *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 2004.
- [17] C. E. L. Peixoto, J. L. N. Audy and R. Prikladinicki, "Effort estimation in global software development projects: Preliminary results from a survey," In Proceedings of International Conference on Global Software Engineering, IEEE Computer Society, 2010, pp. 123-127.
- [18] K. Korhonen and O. Salo, "Exploring quality metrics to support defect management process in a multi-site organization - A case study," In Proceedings of 19th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2008, pp. 213-218.
- [19] M. Bourgault, E. Lefebvre, L. A. Lefebvre, R. Pellerin and E. Elia, "Discussion of metrics for distributed project management: Preliminary findings," In Proceedings of the 35th Annual Hawaii International Conference on System Sciences HICSS'02, IEEE, 2002, 10 p.
- [20] S. Misra, "A metric for global software development environment," In Proceedings of the Indian National Science Academy 2009, pp. 145-158.
- [21] R. M. Lotlikar, R. Polavarapu, S. Sharma and B. Srivastava, "Towards effective project management across multiple projects with distributed performing centers," In Proceedings of IEEE International Conference on Services Computing (CSC'08), IEEE, 2008, pp. 33-40.
- [22] M. T. Lane and P. J. Ågerfalk, "Experiences in global software development - A framework-based analysis of distributed product development projects," In Proceedings of the Fourth IEEE International Conference on Global Software Engineering (ICGSE 2009). 2009, pp. 244-248.
- [23] A. Piri and T. Niinimäki, "Does distribution make any difference? quantitative comparison of collocated and globally distributed projects," In Proceedings of the Sixth IEEE International Conference on Global Software Engineering Workshop (ICGSEW'11), 2011, pp. 24-30.
- [24] B. Sengupta, S. Chandra and V. Sinha, "A research agenda for distributed software development," In Proceedings of the 28th International Conference on Software Engineering, ACM, 2006, pp. 731-740.
- [25] N. Ramasubbu and R. K. Balan, "Globally distributed software development project performance: An empirical analysis," In Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE '07), ACM, 2007, pp. 125-134.
- [26] D. B. Simmons, "Measuring and tracking distributed software development projects," In Proceedings the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003). IEEE, 2003, pp. 63-69.
- [27] D. B. Simmons and N. K. Ma, "Software engineering expert system for global development," In Proceedings of 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), IEEE, 2006, pp. 33-38.
- [28] I. A. da Silva, M. Alvim, R. Ripley, A. Sarma, C. M. L. Werner and A. van der Hoek, "Designing software cockpits for coordinating distributed software development," In the First Workshop on

- Measurement-Based Cockpits for Distributed Software and Systems Engineering Projects, 2007, pp. 14-19.
- [29] E. Carmel, *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.
- [30] E. Carmel and P. Tija, *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce*. Cambridge University Press, the United Kingdom, 2005.
- [31] D. E. Damian and D. Zowghi, "An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations," In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, 2003, 10 p.
- [32] J. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Transactions on Software Engineering*, vol. 29, (6), 2003, pp. 481-494.
- [33] M. Paasivaara and C. Lassenius, "Collaboration practices in global inter-organizational software development projects," *Software Process: Improvement and Practice*, vol. 8, (4), 2003, pp. 183-199.
- [34] R. Battin, R. Crocker, J. Kreidler and K. Subramanian, "Leveraging resources in global software development," *IEEE Software*, vol. 18, (2), 2001, pp. 70-77.
- [35] D. M. Wahyudin, S. Heindl, A. Biffel and B. R. Schatten, "In-time project status notification for all team members in global software development as part of their work environments," In *Proceeding of SOFPIT Workshop 2007, SOFPIT/ICGSE, Munich, 2007*, pp. 20-25.
- [36] J. D. Herbsleb and D. Moitra, "Global software development," *IEEE Software*, vol. 18, (2), 2001, pp. 16-20.
- [37] R. Welborn and V. Kasten, *The Jericho Principle, how Companies use Strategic Collaboration to Find New Sources of Value*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2003.
- [38] H. Holmstrom, E. O. Conchuir, P. J. Ågerfalk and B. Fitzgerald, "Global software development challenges: A case study on temporal, geographical and socio-cultural distance," In *Proceedings of IEEE International Conference on Global Software Engineering (ICGSE'06)*, IEEE, 2006, pp. 3-11.
- [39] V. Casey and I. Richardson, "Virtual teams: Understanding the impact of fear," *Software Process Improvement and Practice*, vol. 13, (6), 2008, pp. 511-526.
- [40] B. Al-Ani and D. Redmiles, "Trust in distributed teams: Support through continuous coordination," *IEEE Software*, vol. 26, (6), 2009, pp. 35-40.
- [41] G. Borchers, "The software engineering impacts of cultural factors on multicultural software development teams," In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, IEEE, 2003, pp. 540-545.
- [42] G. Hofstede, *Culture's Consequences. Comparing Values, Behaviors, Institutions, and Organizations, Across Nations*. Sage Publications, London, 2nd edition, 2001.
- [43] K. H. Möller and D. J. Paulish, *Software Metrics: A Practitioner's Guide to Improved Product Development*. Institute of Electrical & Electronics Engineer, London, 1993.
- [44] CMMI, "CMMI for development," Tech. Rep. version 1.2., Technical Report CMU/SEI-2006-TR-008, 2006.
- [45] W. A. Shewhart, *Statistical Method from the Viewpoint of Quality Control*. Graduate School of Agriculture, Washington, 1939. Referenced in W.E. Deming: *Out of Crisis*. Cambridge, Mass.: MIT Center for Advanced Engineering Study, 1986.
- [46] R. S. Kaplan and D. P. Norton, "The balanced scorecard-measures that drive performance," *Harvard Business Review*, (No. 92105), 1992, pp. 71-79.
- [47] G. Lawrie and I. Cobbold, "Third-generation balanced scorecard: Evolution of an effective strategic control tool," *International Journal of Productivity and Performance Management*, vol. 53, (7), 2004, pp. 611-623.
- [48] D. Card, "Integrating practical software measurement and the balanced scoreboard," In *Proceedings of the 27th Annual International Computer Software and Applications Conference COMPSAC 2003*, 3-6 Nov. 2003, pp. 362- 363.
- [49] PRISMA, *Productivity in Collaborative Systems Development*, ITEA project (2008-2011) number 07024, Project info page, URL: <http://www.itea2.org/project/index/view/?project=237> (Accessed 19.12.2012).
- [50] J. Eskeli, J. Maurologoitia and C. Polcaro, "PSW: A framework-based tool integration solution for global collaborative software development," In *Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA'11)*, Barcelona, Spain, 2011, pp. 124-129.
- [51] I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*. Addison-Wesley Pub Co, Addison-Wesley Object Technology Series, 1999.
- [52] P. Kruchten, "A rational development process," *CrossTalk*, vol. 9, (7), 1996, pp. 11-16.
- [53] E. Demirors, G. Sarmasik and O. Demirors, "The role of teamwork in software development: Microsoft case study," In *Proceedings of the 23rd EUROMICRO Conference, New Frontiers of Information Technology*, 1997, pp. 129-133.
- [54] M. Hoegl and H. G. Gemuenden, "Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence," *Organization Science*, vol. 12, (4), 2001, pp. 435-449.
- [55] ITEA 2, *Information Technology for European Advancement*, ITEA 2 homepage, URL: <http://www.itea2.org/> (Accessed 19.12.2012).
- [56] Tekes, the Finnish Funding Agency for Technology and Innovation, Tekes homepage. URL: <http://www.tekes.fi/eng/> (Accessed 19.12.2012).

Quality-Oriented Design of Software Services in Geographical Information Systems

Michael Gebhart

Gebhart Quality Analysis (QA) 82
Karlsruhe, Germany
michael.gebhart@qa82.de

Suad Sejdovic

Campana & Schott
Stuttgart, Germany
suad.sejdovic@campana-schott.com

Abstract—Distributed information, such as sensor information, increasingly constitutes the basis for geographical information systems. For that reason, these systems are designed according to services-oriented design principles, which means that they require software services returning necessary information and provide higher-value ones. These services are expected to follow quality attributes, such as loose coupling and autonomy, which have been identified as important in the context of service-oriented architectures. For measuring these quality attributes, metrics have been derived that enable quantifications. They can be directly evaluated on basis of formalized service designs and indicate the extent of quality attributes. This article shows the application of these service design metrics for a quality-oriented design of services in geographical information systems. The considered system is part of the Personalized Environmental Service Configuration and Delivery Orchestration project of the European Commission.

Keywords—service; design; quality; geographical information system; case study

I. INTRODUCTION

A geographic information system (GIS) is a computer system, which is used for capturing, storing, analyzing and also displaying geospatial data, whereas geospatial data is data that is describing characteristics of spatial features on the Earth's surface which are referenced to by a location [1].

In order to access this data in a standardized manner, it is provided by means of software services that base on standardized protocols and interface description languages, such as Simple Object Access Protocol (SOAP) over HyperText Transfer Protocol (HTTP) and Web Services Description Language (WSDL) [2]. Besides the usage of services, the information systems themselves are often required to be integrated in a more complex architecture. This is why the systems are additionally supposed to not only invoke but also to provide services that enable accessing higher-value functionality. As result, geographical information systems apply services as architecture paradigm and follow service-oriented design principles.

In the context of service-oriented architectures (SOA) several quality attributes have been identified as important depending on higher-level quality goals that are associated with the system. In order to easily switch between several data sources, for geographical systems a very important

aspect is to build a flexible and maintainable architecture. These higher-level quality goals can be broken down into more fine-grained quality attributes, such as loose coupling and autonomy, affecting the building blocks of the architecture, in this case the services. Accordingly, the used services in the context of the geographical information system have to be designed in a way that these quality attributes can be fulfilled. The design of services can be confined to a service interface and a service component. Whilst the service interface describes the externally visible access point to the service, the service component focuses on the internal behavior of the service itself. In order to formalize the design of a service, the Service oriented architecture Modeling Language (SoaML) as profile for the Unified Modeling Language (UML) can be applied [3]. It represents an emerging standard to describe service designs in a standardized manner and gains increasing tool support, which leads to an increasing acceptance in development processes.

For measuring the quality of software, metrics can be used as quantified values of quality indicators [4], [5], [6], [7], [8]. In the context of service-oriented architectures and in particular for the design of services, Gebhart et al. identified metrics especially evaluating service designs based on the Service oriented architecture Modeling Languages (SoaML) [4]. These ones refer to model elements available within this Unified Modeling Language (UML) profile, which simplifies the evaluation of formalized service designs. Compared to other non-formalized quality indicators, such as textual descriptions, or metrics not designed for SoaML, the usage of these SoaML-oriented metrics avoids interpretation effort with possibly faulty interpretation and accordingly faulty measurement. Finally, the metrics can be automatically calculated as implemented by the QA82 Architecture Analyzer [9].

In order to demonstrate the quality-oriented design of services based on these metrics, this article considers the design of a geographical information system in a service-oriented manner [1]. This means that metrics especially designed for service designs based on SoaML are applied for designing services of a geographical information system with certain quality attributes fulfilled. In this article, the project Personalized Environmental Service Configuration and Delivery Orchestration (PESCaDO) is considered [10].

The article is organized as follows: Section II introduces the service design process, the formalization of service designs using SoaML, and wide-spread quality attributes. The scenario is introduced in Section III and in Section IV the services are designed. Section V concludes this article and introduces future research work.

II. BACKGROUND

This section describes fundamentals for the article. This includes especially the understanding of service designs in the context of software service engineering and its formalization using SoaML.

A. Service Design Process

The service design phase is a primary ingredient of the software service engineering that can be understood as the “discipline for development and maintenance of SOA-enabled applications” [11]. The central purpose of the service design phase is to create a formalized draft of services, so-called service designs, before implementing them. This enables the adaptation and optimization of the entire services architecture without cost-intensive source code changes. That is why analyses of the designs regarding quality attributes, such as loose coupling, are required to be performed within the service design phase. In [12], Gebhart introduces a service design process reusing existing work of Erl, IBM et al. [13], [14], [15], [16], [17], [18] and describes necessary steps within the service design phase for fulfilling this requirement. Figure 1 illustrates this process.

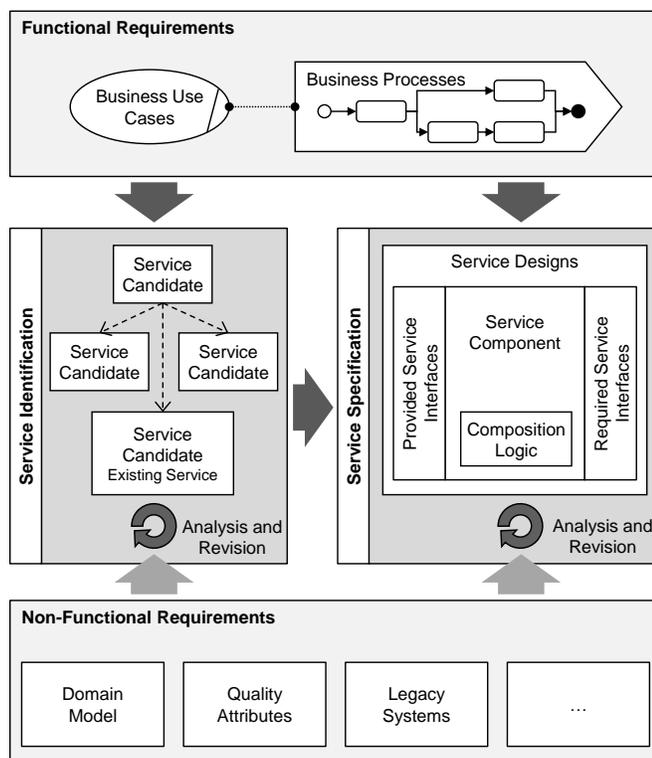


Figure 1. Quality-oriented service design process.

The service design process is a combination of systematic derivations and subsequent analyses and revisions. The systematic derivation especially considers the fulfillment of functional requirements that have been identified within the requirements analysis phase: In a first step, the functional requirements are transferred into so-called service candidates. These represent preliminary services that are not fully specified yet [13]. Especially, when existing services have to be taken into account, there is no necessity to specify new service designs. Instead, the existing specifications can be reused. Otherwise, the service candidates are transferred into elements of service designs. For example, for each service candidate a service interface and implementing service component is created.

The iterative analysis and revision focuses on the fulfillment of non-functional requirements, such as quality attributes. Within each iteration first the current state is analyzed regarding non-functional requirements. For example, the quality attributes are determined using appropriate metrics as demonstrated by Gebhart et al. in [19]. Afterwards, the artifacts are revised for improving the quality attributes or other non-functional requirements. As a result, service designs are created that both fulfill functional requirements that have been determined within the requirements analysis phase and non-functional ones, such as loose coupling, that support higher-level quality goals.

The created service designs can be used to derive web service implementation artifacts in a model-driven way as introduced by Hoyer et al. in [20] and Gebhart et al. in [21].

B. Service Design Formalization

For formalizing a service design, in this article SoaML is applied [3]. In comparison to other proprietary languages, such as the UML Profile for Software Services developed by IBM [22], SoaML is a profile for UML [23] and a metamodel standardized by the Object Management Group (OMG). It provides elements necessary to describe service-oriented architectures and its building blocks, the services. In the meanwhile, SoaML is an emergent standard adopted by several tool vendors. Even IBM has replaced its proprietary UML profile with SoaML [24]. In this article SoaML is applied as UML profile.

In order to model service designs with SoaML, necessary elements of the profile have to be identified. This article uses the elements as introduced by Gebhart et al. in [25]. The service design formalization consists of both the formalization of service candidates and service designs. Thus, for both sub-phases of the service design phase the adequate formalization has to be determined.

According to Erl [26], a service candidate represents a preliminary service on a high level of abstraction. During this phase, only possible operations, called operation candidates, service candidates as grouping of these capabilities, and dependencies between service candidates are determined. In SoaML the Capability element exists, which corresponds to this understanding. The following table shows the mapping of service candidate elements on a conceptual level onto elements within SoaML.

TABLE I. MAPPING BETWEEN SERVICE CANDIDATE ELEMENTS AND SOAML

Service Candidate Element	SoaML Element
Service Candidate	Capability (UML class that is stereotyped with "Capability")
Operation Candidate	Operation within a Capability element
Dependency	Usage Dependency between Capability elements

This table demonstrates that there is a one-to-one mapping between service candidate elements and elements within SoaML possible. Figure 2 illustrates the modeling of service candidates in SoaML.

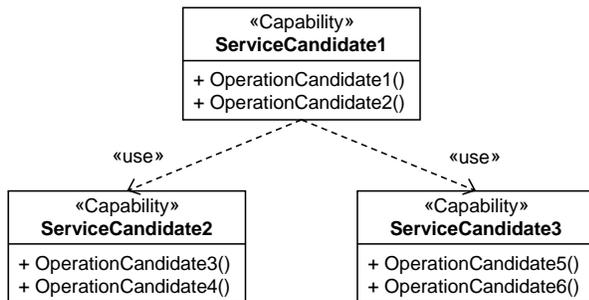


Figure 2. Service candidates in SoaML.

The example includes three service candidates with each of them containing two operation candidates. In this case ServiceCandidate1 requires operations of ServiceCandidate2 and ServiceCandidate3 for fulfilling its functionality.

TABLE II. MAPPING BETWEEN SERVICE DESIGN ELEMENTS AND SOAML

Service Design Element	SoaML Element
Service Interface	ServiceInterface (UML class that is stereotyped with "ServiceInterface")
Provided Operation	Operation within an interface that is realized by the ServiceInterface element
Realized Operation	Operation within an interface that is associated with the ServiceInterface by using a Usage Dependency in UML
Role	Property within the ServiceInterface that is typed by the interface that contains the provided operations or by the interface that contains the required operations
Interaction Protocol	A behavior, such as an UML Activity
Service Component	Participant (UML component that is stereotyped with "Participant")
Provided Service	Service (UML Port that is stereotyped with "Service")
Required Service	Request (UML Port that is stereotyped with "Request")
Internal Behavior	UML Activity that is added as OwnedBehavior to the Participant

A service design represents a full specification of a service [27]. It includes both the service interface as externally visible access point and the service component as realization of the business logic. The service interface has to specify the operations provided by the service and the ones required in order to receive callbacks. Additionally, the participating roles and the interaction protocol have to be determined. Latter describes in which order the operations have to be called for obtaining a valid result.

The service component consists of the services provided by the component and the ones required by the component for fulfilling its functionality. Additionally, the internal behavior is specified by means of a flow of activities that is the composition in case of a composed service. In SoaML there exist elements that directly correspond to the described understanding.

Table II shows the mapping according to [27]. Whilst the original work bases on SoaML in version 1.0 Beta 1, the table was adapted that it corresponds to the standard in the current version 1.0 final.

To illustrate the modeling of service designs, the following figures illustrate the modeling of a service interface and a service component in SoaML. The service interface in Figure 3 assumes two participants interacting, the provider and the consumer. The provider offers two operations the consumer can call. Furthermore, also the consumer has to provide one operation for receiving callbacks. The interaction protocol describes the operation call order for a valid result.

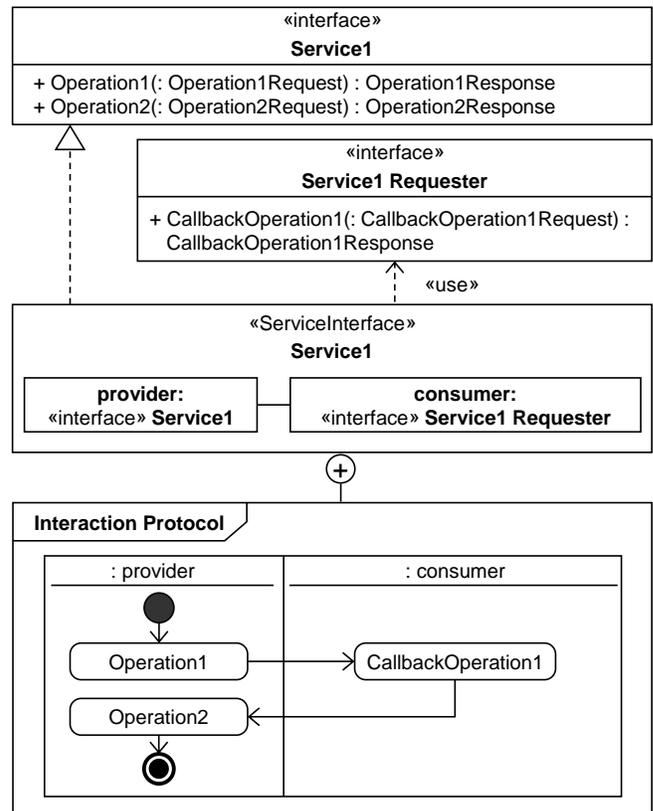


Figure 3. Service interface in SoaML.

The service component illustrated in Figure 4 provides one service and requires one service for fulfilling its functionality. It consists of two internal components, one realizing the composition logic and one implementing further internal logic. The internal behavior can be described by means of an owned behavior in UML. For the sake of simplicity, the internal behavior is not illustrated.

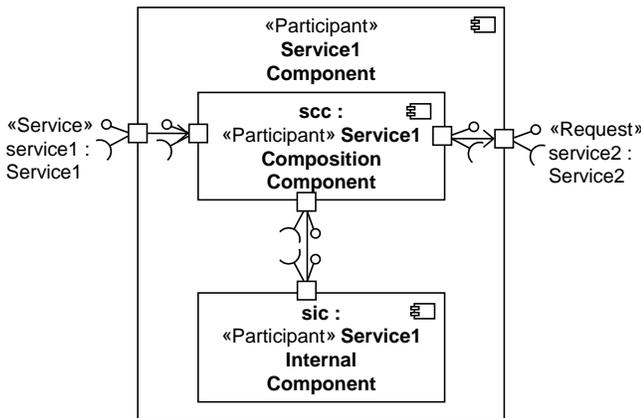


Figure 4. Service component in SoaML.

C. Quality Attributes and Metrics

With the establishment of service-oriented architectures, several strategic goals are associated. Examples are the higher flexibility of the architecture and its easier maintenance [28]. In order to fulfill these strategic goals, quality attributes, such as loose coupling and autonomy, for the building-blocks of the architecture, the services, have been identified. The fulfillment of these quality attributes provides the basis for achieving the strategic goals. As these quality attributes yet are described on an abstract level, they can be further broken down into measurable quality indicators that refer to concrete elements of the services [25]. If these elements are described during design time, the quality indicator can be determined on basis of a service design model. A metric describes the formula for a certain quality indicator and enables its concrete quantification.

In [27], Gebhart et al. identified quality attributes for services that are considered as important in this context. Quality indicators and metrics that enable their determination on basis of formalized service designs are derived in [4]. Based on this work, this article uses the following quality attributes and quality indicators.

1) *Unique Categorization*: The first quality attribute is the unique categorization, which is comparable to cohesion. According to its description, a service should provide functionality that belongs together. In literature the categorization is mostly described by means of service categories, such as entity, task, and utility services [29]. The quality attribute can be described in detail by means of quality indicators:

First, technical and business-related functionality should be separated up into two services. As technical functionality is used by a different target group than business-related one

this helps to maintain the services. This corresponds to the distinction between entity / task services and utility services as introduced by Erl [13], [30].

In order to further increase the maintainability of services also functionality that can be reused in several contexts, i.e., general one or also known as agnostic, should be separated from specific one [26]. This encourages the reuse of general functionality and avoids the influence of changes concerning specific functionality on the general and highly used one. This results in a distinction between entity services that provide general and entity-based operations and task services with mostly specific operations [29]. However, whether functionality is agnostic or not depends on personal estimation.

According to the data superiority, when a service manages a business entity, it should be the only one. This is important to avoid redundant functionality within various services. For the categorization this means that there are no entity services for the same business entity.

Finally, all operations within one service should work on the same business entities. This means that within all operations the same business entity is used. As result, this quality indicator measures whether an entity service is managing only one business entity as expected for an entity service.

2) *Discoverability*: The best service cannot be leveraged when it cannot be found. That is why discoverability is an important aspect concerning the reusability of services [30]. The discoverability as quality attribute can be refined by the following quality indicators:

First, services and operations should have functional names. Only in this case a service and the contained operations can be found.

In order to increase this aspect, the naming should follow known naming conventions. This can be both the language of the artifacts and the case sensitivity. Also other rules, such as naming operations by using a verb and a noun, are often applied [31].

Finally, the more information is provided the faster a service can be found. This means that especially when modeling services, as most information as possible should be given.

3) *Loose Coupling*: One of the most often referred quality attribute is loose coupling. It focuses on the dependencies between services, which influences the flexibility and maintainability of services. The following quality indicators that are measurable on service designs can be identified:

In order to support long-running operations, these operations should be provided asynchronously. This means that if an operation provides a long-running functionality an appropriate callback operation should be provided by the consumer and invoked when the operation is finished. This enables the exchange of service provider and consumer during the operation execution.

The dependency between services is also influenced by commonly used data types. Especially when services commonly use complex data types they are dependent as changing one data type requires changes within all using

services. The loose coupling can be measured by the degree to which complex data types are commonly used. Best, services share only simple types. Of course, services can work on the same business entity, such as a Person entity, however the data types should be only copies. A canonical data schema as part of an enterprise service bus should map similar or identical data.

To further increase the independence between services the operations and parameters should be abstract. This means that no technical background information should be necessary to use a service [31]. Also parameters should not include technical data types. This supports the exchange of services as the implementation details are hidden.

If an operation provides functionality resulting in state changes there should be always a compensating undo operation. This again reduces the dependency between services.

4) *Autonomy*: Finally, the autonomy is one of the considered quality attributes. It also considers the dependency between services but focuses on the ability of a service to be used without other services.

The first quality indicator considers the direct dependency between services, i.e., how many other services are required for fulfilling the own functionality. Basic services are mostly highly autonomous. Composite services instead are composing existing functionality and are thus not autonomously usable.

The second quality indicator focuses on the functional overlap between services. If the functionality of a service overlaps with the one of other services, in most cases the service can also be only used together with the other ones, because in most scenarios functionality of all these services is required. Thus, even though there is no direct dependency between the services, because of the overlapping functionality the service cannot be used solely.

III. SCENARIO

This section introduces the underlying scenario for the exemplary quality-oriented design of software services in this article, the project Personalized Environmental Service Configuration and Delivery Orchestration (PESCaDO) of the European Commission (EC) [10], [32]. The overall goal of the system is to assist human beings in decision-finding under consideration of the personal profile. For example, a user with a pollen allergy and heart problems at very high temperatures wants to know, whether it is advisable for him to book a bicycle tour within the next few months. As described in [1], one special requirement is the semantic support for accessing environmental data. Thus, the system should be capable to identify any related data sources for a requested phenomenon like pollen. That is, the system has to be able to extend a single requested phenomenon by other more specific related ones, like “Birch Pollen”.

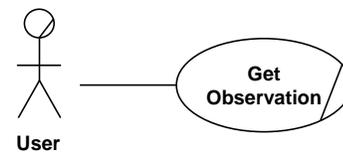


Figure 6. Considered business use case.

Regarding PESCaDO the business use case in Figure 6 can be identified. The business use case describes the requirement to get an observation, which results in a value describing some phenomenon. It is modeled using the adapted notation for use case diagrams by the UML profile for business modeling as introduced by IBM [33], [34]. It is very important to achieve a deep understanding about the business use case, as it is the basic artifact for the identification of service candidates in the service design

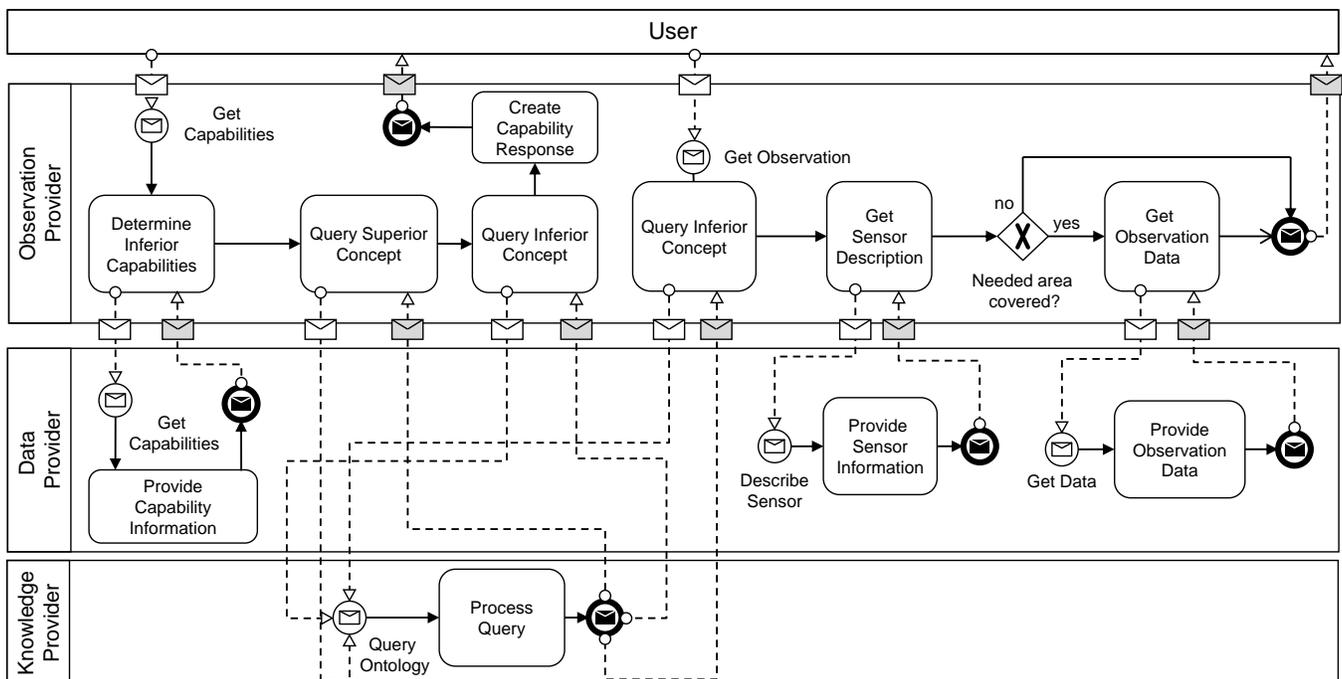


Figure 5. Considered business process.

phase. Thus, knowledge about the internal behavior of the business use case is important within the service design process. This internal behavior can be modeled using the Business Process Model and Notation (BPMN) [35], whereby the modeling concentrates on activities that could be processed automatically.

Figure 6 shows the business process that covers the data access under consideration of the semantic information that is given within the request. The BPMN model consists of four pools. The first pool, labeled with “User” represents the user and is collapsed, as the contained activities are not relevant for automation. The three remaining pools are expanded, as they contain relevant activities for further steps in the development process. Interactions between the different providers are shown by message flows between the pools, whereas the message flows are representing requests and the resulting answers. As the business process is a fundamental artifact, it has to be clear and unambiguous before entering the service design phase.

The observation provider offers two functionalities to the user. The first functionality “Get Capabilities” refers to the capabilities of the observation provider. It represents the self-description capability of the service. By requesting the capabilities the user initiates a procedure, which dynamically generates the information about the capabilities with regard to the underlying data sources. For this, the capabilities of the underlying data provider have to be requested. The data provider also offers the self-describing functionality “Get Capabilities”, which returns information about its functionality and the type of data that is available. The information about the available data is returned as a concept referring to the content in an ontology [36], [37]. The returned data can now be used within the observation provider to generate a semantic hierarchy by gaining details about the inferior and superior concepts of the retrieved concept. All the required data is provided by the knowledge provider, which knows all relevant concepts and relationships between them. An important functionality to enable such hierarchies is the functionality “Query Ontology”. Through this functionality it is possible to query the ontology and determine the required information. For instance, a data source may contain information about birch pollen and refers as a consequence to a concept called “BirchPollen” within the ontology. The knowledge provider may now generate a hierarchy, which is presenting the position of this concept within a hierarchy, if one exists. For example, the knowledge provider may return a relationship between the concepts “Pollen” and “BirchPollen”. Thus, a request for data containing information about the concept “Pollen” should also take into consideration any data about the inferior concept “BirchPollen”. This feature supports the requester to find information for more complex concepts, which are referring to composite phenomena, such as air quality.

After retrieving all necessary information, the observation provider processes all retrieved data and generates the requested reply. Thus, the user gets a structured, hierarchical view on the available data. This

dynamic approach ensures that users can always get a current view on all available information.

The second functionality of the observation provider, “Get Observation” realizes the data access, whereas the request is addressed to the knowledge provider to determine the inferior concepts of the user input. Thus, all relevant data is found and returned. The next step is to verify that any relevant data is also available for the given area and/or date before requesting the quality parameters from the data provider. The quality parameters give some indication of the quality and accuracy of the available data. Within the last step, all available information is retrieved and delivered to the user.

IV. QUALITY-ORIENTED SERVICE DESIGN

In this section, the services for the described scenario are designed considering the quality attributes introduced in the Background section. For this purpose, first service candidates are systematically derived from the business requirements. Afterwards, these candidates are analyzed and revised according to the quality attributes. The revised service candidates are used to derive service designs as full specifications of the required services. Finally, the service designs are again analyzed and revised. As result, service designs are created that fulfill both the functional requirements and certain quality attributes.

A. Derivation of Service Candidates

In a first step, service candidates have to be derived from the modeled business requirements. This step can be performed systematically, as there exist clear descriptions about which elements are transformed into which ones. For this step especially the business process has to be considered as it describes provided functionality and the dependencies between participating roles. Figure 7 shows the methodology for service candidate derivation.

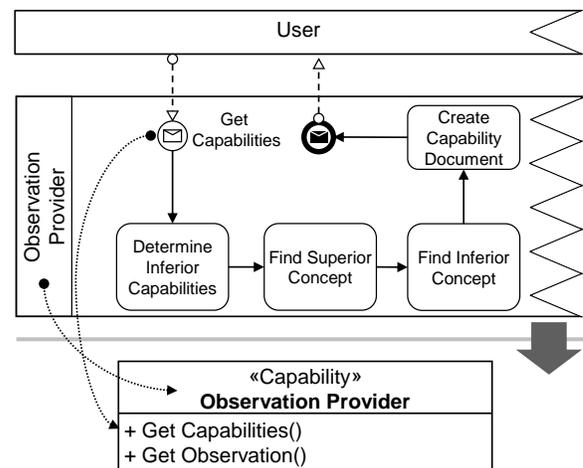


Figure 7. Derivation of service candidates.

Each pool is transformed into a service candidate and each message start event that represents an available operation is transformed into an operation candidate.

The dependencies between the service candidates are derived from the operation calls between the pools. As result three service candidates can be derived as shown in Figure 8.

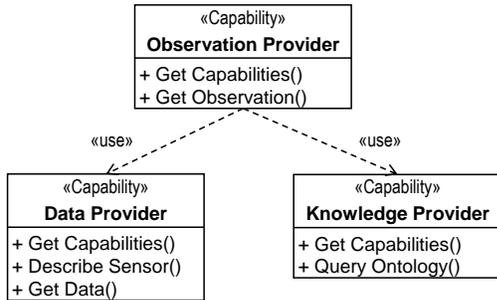


Figure 8. Derived service candidates.

B. Service Candidate Analysis and Revision

In order to assure a high quality of the services already during this phase, a quality analysis is performed. For that purpose, the service candidates are analyzed by measuring the quality indicators introduced in the Background section. During this phase not all quality indicators are applicable as some information might be missing. Based on the available information, the following quality indicators are determined. The used metrics are taken from Gebhart et al. [4].

1) *Unique Categorization*: In order to measure the separation of technical and business-related functionality, the following metric is applied.

$$DBTF(sc) = \frac{|BF(OC(sc))|}{|OC(sc)|}$$

TABLE III. VARIABLES AND FUNCTIONS USED FOR DBTF

Element	Description
DBTF	Division of Business-related and Technical Functionality
sc	service candidate: the considered service candidate
s	service: the considered service that is provided or required, represented by an ServicePoint or RequestPoint in SoaML
BF(oc)	Business-related Functionality: operation candidates providing business-related functionality out of the set of operation candidates oc
BF(o)	Business-related Functionality: operations providing business-related functionality out of the set of operations o
OC(sc)	Operation Candidates: operation candidates of the service candidate sc
SI(s)	Service Interface: service interface of the service s. In SoaML it is the type of the ServicePoint or RequestPoint s
RI(si)	Realized Interfaces: realized interfaces of the service interface si
O(i)	Operations: operations within the interface i
oc	Number of operation candidates oc
o	Number of operations o

As all service candidates were derived from the business process they provide business-related functionality only. The value of DBTF for all service candidates is 1. The following table shows the interpretation of this value.

TABLE IV. INTERPRETATION OF VALUES FOR DBTF

Value	Interpretation
0	Only technical functionality is provided
Between 0 and 1	Both business-related and technical functionality is provided
1	Only business-related functionality is provided

This table acknowledges that only business-related functionality is provided. As 0 and 1 are desired values, all service candidates fulfill this aspect optimally. The next quality indicator measures the separation of agnostic and non-agnostic functionality, i.e., the separation of general and highly specific operations. The following metric is applied.

$$DANF(sc) = \frac{|AF(OC(sc))|}{|OC(sc)|}$$

TABLE V. VARIABLES AND FUNCTIONS USED FOR DANF

Element	Description
DANF	Division of Agnostic and Non-agnostic Functionality
AF(oc)	Agnostic Functionality: operation candidates providing agnostic functionality out of the set of operation candidates oc
AF(o)	Agnostic Functionality: operations providing agnostic functionality out of the set of operations o

The determination whether an operation provides agnostic functionality or not requires personal estimation. As all operations are generally named and provide functionality that is not specific to a certain scenario, they are assumed as agnostic. As result the metric returns 1 for all service candidates. According to the following table, this represents the case that only agnostic functionality is provided.

TABLE VI. INTERPRETATION OF VALUES FOR DANF

Value	Interpretation
0	Only non-agnostic functionality is provided
Between 0 and 1	Both agnostic and non-agnostic functionality is provided
1	Only agnostic functionality is provided

Also in this case the values 0 and 1 are desired for a unique categorization. Accordingly, a revision regarding this quality indicator is not necessary. For measuring the data superiority the following metric is applied.

$$DS(sc) = 1 - \frac{|MBE(OC(sc)) \cap MBE(OC(ALL_{sc} \setminus sc))|}{|MBE(OC(sc))|}$$

TABLE VII. VARIABLES AND FUNCTIONS USED FOR DS

Element	Description
DS	Data Superiority
M1 \ M2	Elements of set M1 without elements of set M2 or the element M2
ALL _{sc}	All existing service candidates
ALL _s	All existing services
MBE(oc)	Managed Business Entities: business entities that are managed by operation candidates oc
MBE(o)	Managed Business Entities: business entities that are managed by operations o

In order to determine the results, the service candidates have to be inspected in detail. All service candidates do not manage business entities as it is known in a typical business environment. In this case, a more data-centric view is required that can be mapped onto the quality indicator.

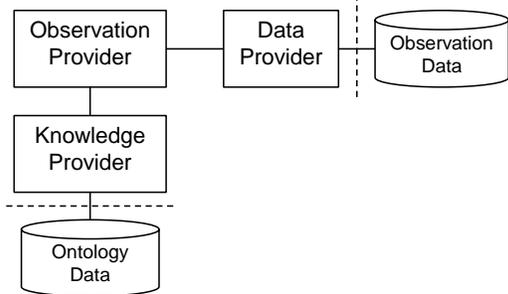


Figure 9. Accessed data storages.

Figure 9 illustrates the services and their access to data storages. This shows that the Data Provider accesses observation data and the Knowledge Provider manages ontology data. The Observation Provider is not responsible for any data directly. As result, for each service candidate but Observation Provider the metric returns 1, which represents the desired value. For Observation Provider this metric is not defined. To exemplify the calculation, the following formula demonstrates it for the Knowledge Provider.

$$DS(Knowledge Provider) = 1 - \frac{| \{Ontology Data\} \cap \{Observation Data\} |}{| \{Ontology Data\} |} = 1 - 0 = 1$$

TABLE VIII. TEXT INTERPRETATION OF VALUES FOR DS

Value	Interpretation
Less than 1	No data superiority regarding the managed business entities
1	Data superiority regarding the managed business entities

As result, also in this case there is no revision necessary as all service candidates fulfill the unique categorization concerning this quality indicator optimally. The usage of common business entities can be measured using the following metric.

$$CBEU(sc) = \frac{OCUBE \left(OC(sc), CMP \left(OC(sc), MOUBE(OC(sc)), UBE(OC(sc)) \right) \right)}{| OC(sc) |}$$

TABLE IX. VARIABLES AND FUNCTIONS USED FOR CBEU

Element	Description
CBEU	Common Business Entity Usage
CMP(oc, be1, be2)	Composition: biggest set of business entities managed by operation candidates oc out of be2 that depend on business entities be1
CMP(o, be1, be2)	Composition: biggest set of business entities managed by operations o out of be2 that depend on business entities be1
UBE(oc)	Used Business Entities: business entities that are used within operation candidates oc as input
UBE(o)	Used Business Entities: business entities that are used within operations o as input
MOUBE(oc)	Mostly Often Used Business Entities: business entities that are mostly often used within one operation candidate out of operation candidates oc
MOUBE(o)	Mostly Often Used Business Entities: business entities that are mostly often used within one operation out of operations o
OCUBE(oc, be)	Operation Candidates Using Business Entities: operation candidates out of operation candidates oc that only use business entities out of be
OUBE(o, be)	Operations Using Business Entities: operations out of operations o that only use business entities out of be

The calculation of this metric is exemplified for Observation Provider that does not use business entities in any of its operation candidates. In order to comprehend the calculation every function within the formula is calculated separately.

$$OC(Observation Provider) = \{Get Capabilities, Get Observation\}$$

$$MOUBE(OC(Observation Provider)) = \{ \}$$

$$CMP(\dots) = \{ \}$$

$$OCUBE(\dots) = \{Get Capabilities, Get Observation\}$$

$$CBEU(Observation Provider)$$

$$= \frac{| \{Get Capabilities, Get Observation\} |}{| \{Get Capabilities, Get Observation\} |} = 1$$

Summarized, every service candidate uses in all of its operation candidates the same business entity and thus is only responsible for one certain business entity or parts of it. Also in this case there is no revision necessary. Thus, the service candidates fulfill the unique categorization optimally.

2) *Discoverability*: As service candidates describe services and their dependencies in an abstract manner, the discoverability is only important for service designs. Thus, the discoverability will not be measured during this phase but later during the analysis and revision of service designs.

3) *Loose coupling*: In order to measure the asynchrony for long-running operations details of the service designs are necessary. During the specification of service designs it is determined whether an operation is provided synchronously or asynchronously. Similarly, the complexity of common data types can only be determined when the data types are specified. Thus, also this aspect cannot be measured on service candidates but only on service designs. As provided operations are not final yet and parameters are not defined also the abstraction cannot be measured.

The only quality indicator measurable on basis of service candidates is the compensation. For that purpose, the following metric can be applied.

$$CF(sc) = \frac{|CFP(SC(NC(OC(sc))))|}{|SC(NC(OC(sc)))|}$$

TABLE X. VARIABLES AND FUNCTIONS USED FOR CF

Element	Description
CF	Compensating Functionality
NC(oc)	Non-Compensating: non-compensating operation candidates out of the set of operation candidates oc
NC(o)	Non-Compensating: non-compensating operations out of the set of operations o
SC(oc)	State Changing: operation candidates out of the set of operation candidates oc that provide state-changing functionality
SC(o)	State Changing: operations out of the set of operations o that provide a state-changing functionality
CFP(oc)	Compensating Functionality Provided: operation candidates out of the set of operation candidates oc a compensating operation candidate exists for
CFP(o)	Compensating Functionality Provided: operations out of the set of operations o a compensating operation exists for

TABLE XI. INTERPRETATION OF VALUES FOR CF

Value	Interpretation
Less than 0	There exist state-changing operation candidates respectively operations without compensating operations candidates respectively operations
1	For all operation candidates respectively operations that provide state-changing functionality a compensating operation candidate respectively operation exists

As the Observation Provider only returns information and does not change the state of any artifact, the metric is not defined and there is no revision necessary. Otherwise, the table above lists the values and their interpretation.

4) *Autonomy*: The dependencies between services can be measured on basis of service candidates using the following metric.

$$SD(sc) = |RS(sc)|$$

TABLE XII. VARIABLES AND FUNCTIONS USED FOR SD

Element	Description
SD	Service Dependency
RS(sc)	Required Services: service candidates the service candidate sc depends on
SCT(s)	Service Component: service component of the service s
RS(sct)	Required Services: services the service component sct depends on

For the Observation Provider the metric returns the value 2 as the candidate depends on two other services.

TABLE XIII. INTERPRETATION OF VALUES FOR SD

Value	Interpretation
0	the service candidate or the functionality fulfilling service component depends on no other service candidate respectively service
n (n > 0)	the service candidate or service component requires n other services to fulfill its functionality

Although the value is not optimal, there is no revision possible. The quality indicator shows that there are dependencies, however as the Observation Provider represents a composed service, there is no possibility to improve the quality indicator. Additionally, solving these dependencies would impact other quality indicators, such as those determining the unique categorization.

The functional overlap can be measured using the following metric.

$$FO(sc) = \frac{|OF(OC(sc), OC(ALL_{sc} \setminus sc))|}{|OC(sc)|}$$

TABLE XIV. VARIABLES AND FUNCTIONS USED FOR FO

Element	Description
FO	Functionality Overlap
OF(oc1, oc2)	Overlapping Functionality: operation candidates out of the set of operation candidates oc1 with overlapping functionality to the operation candidates oc2
OF(o1, o2)	Overlapping Functionality: operations out of the set of operations o1 with overlapping functionality to the operations o2

As in case of the Observation Provider there is no functional overlap, the metric returns 0.

As 0 represents the desired value, there is no revision required. Summarized, the service candidates fulfill nearly all quality indicators optimally. Only the autonomy is not optimal, however this quality indicator cannot be improved

without worsen other quality indicators. Additionally, the composition including the dependencies is intended. Nevertheless, the quality indicators points to the fact that we have dependencies that influence the maintainability and flexibility of the architecture. This has to be kept in mind.

TABLE XV. INTERPRETATION OF VALUES FOR FO

Value	Interpretation
0	The operation candidates respectively operations of the considered service candidate or service do not provide functionality that overlaps with functionality of other service candidates or services
Between 0 and 1	The operation candidates respectively operations of the considered service candidate or service provide functionality that overlaps with functionality of other service candidates or services
1	The operation candidates respectively operations of the considered service candidate or service provide only functionality that overlaps with functionality of other service candidates or services

C. Derivation of Service Designs

Subsequent to the service identification, the service specification can be performed.

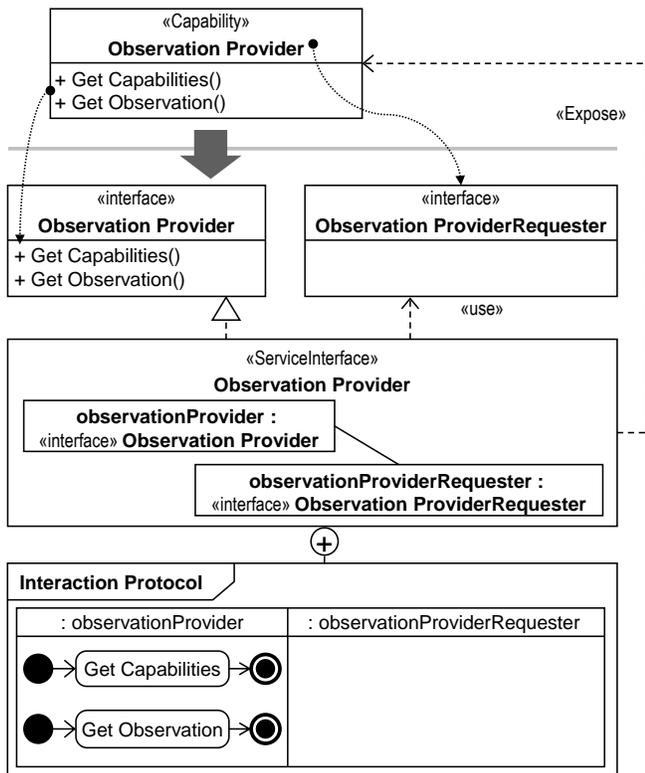


Figure 10. Derivation of service interfaces.

Also in this case, first the service candidates are systematically transformed into service designs. Afterwards, the service designs are iteratively analyzed and revised.

As described in the Background section, a service design consists of a service interface and a service component. Figure 10 illustrates the derivation of a service interface from a service candidate. The service component can be similarly derived as shown in Figure 11.

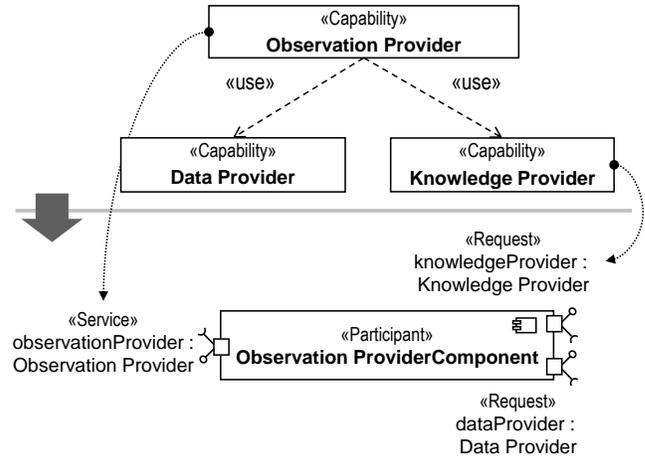


Figure 11. Derivation of service components.

The initial service interfaces and service components are derived from the corresponding capability elements. To create a reference between the service design and the business, the related capability element is attached to the service interface by means of an «Expose» association. The dependencies of the capability elements are reflected with «use» relationships. This relationship information provides the input for the derivation of the ports of the service component. Further details about the systematic derivation are described by Gebhart et al. in [27].

D. Service Design Analysis and Revision

Similarly to the service identification phase, also within the service specification phase an analysis and revision is performed after the systematic derivation of service designs. As the service designs were derived from service candidates with optimized quality indicators, also on basis of service designs most quality indicators will be optimal from the beginning. However, there are some indicators that were not measurable on basis of service candidates. For the sake of completeness, in this section metrics for all quality indicators with focus on service designs are listed. The metrics use the variables and functions introduced above. Also the interpretation of values is identical.

1) *Unique Categorization*: The quality indicators for the unique categorization can be measured by the following metrics. These metrics focus on the specifics of service designs. The first metric measures the division of business-related and technical functionality.

$$DBTF(s) = \frac{|BF(O(RI(SI(s))))|}{|O(RI(SI(s)))|}$$

As service within the formula the service described as UML port within the service component, i.e., the Participant in SoaML, has to be chosen.

The division of agnostic and non-agnostic functionality is measured by the following metric.

$$DANF(s) = \frac{|AF(O(RI(SI(s))))|}{|O(RI(SI(s)))|}$$

Also the data superiority differs only in the methodology how to determine the relevant operations. Compared to the service candidates, the operations within the realized interface of the service interface have to be chosen.

$$DS(s) = 1 - \frac{|MBE(O(RI(SI(s)))) \cap MBE(O(RI(SI((ALL_s \setminus s)))))|}{|MBE(O(RI(SI(s))))|}$$

The common business entity usage can be measured using the following metric.

$$CBEU(s) = \frac{|OUBE\left(\begin{matrix} O(RI(SI(s))), \\ CMP\left(\begin{matrix} O(RI(SI(s))), MOUBE(O(RI(SI(s))), \\ UBE(O(RI(SI(s)))) \end{matrix}\right) \end{matrix}\right)|}{|O(RI(SI(s)))|}$$

As the service designs were derived from high-quality service candidates, all metrics have the same results as on basis of service candidates. So, there is no revision necessary.

2) *Discoverability*: The discoverability could not be measured on service candidates as they only represent abstract services with non-final names. Thus, new metrics have to be introduced.

The functional naming of service interfaces, roles, operations, parameters, and data types are measured by the following metrics.

$$FNSI(s) = \frac{|FN(SI(s))|}{|SI(s)|}$$

$$FNR(s) = \frac{|FN(R(SI(s)))|}{|R(SI(s))|}$$

$$FNO(s) = \frac{|FN(O(RI(SI(s))))|}{|O(RI(SI(s)))|}$$

$$FNP(s) = \frac{|FN(P(O(RI(SI(s)))))|}{|P(O(RI(SI(s))))|}$$

$$FNDT(s) = \frac{|FN(DT(P(O(RI(SI(s))))))|}{|DT(P(O(RI(SI(s))))|}$$

TABLE XVI. VARIABLES AND FUNCTIONS USED FOR FNSI, FNR, FNO, FNP, AND FNDT

Element	Description
FNSI	Functional Naming of Service Interface
FNR	Functional Naming of Roles
FNO	Functional Naming of Operations
FNP	Functional Naming of Parameters
FNDT	Functional Naming of Data Types
FN(me)	Functional Naming: set of functionally named elements out of the set of modelling elements me
P(o)	Parameters: parameters of the operations o and in case of messages the contained parameters
DT(p)	Data Types: used data types (recursively continued) of parameters p
R(si)	Roles: roles of service interface si

As the original service candidates were derived from business requirements the metric always returns 1 with the following interpretation.

TABLE XVII. INTERPRETATION OF VALUES FOR FNSI, FNR, FNO, FNP, AND FNDT

Value	Interpretation
Less than 1	There are elements that are not functionally named
1	All elements are functionally named

The naming convention compliance of service interfaces, roles, operations, parameters, and data types, can be measured as follows:

$$NCCSI(s) = \frac{|NCC(SI(s))|}{|SI(s)|}$$

$$NCCR(s) = \frac{|NCC(R(SI(s)))|}{|R(SI(s))|}$$

$$NCCO(s) = \frac{|NCC(O(RI(SI(s))))|}{|O(RI(SI(s)))|}$$

$$NCCP(s) = \frac{|NCC(P(O(RI(SI(s)))))|}{|P(O(RI(SI(s))))|}$$

$$NCCDT(s) = \frac{|NCC(DT(P(O(RI(SI(s))))))|}{|DT(P(O(RI(SI(s))))|}$$

TABLE XVIII. VARIABLES AND FUNCTIONS USED FOR NCCSI, NCCR, NCCO, NCCP, AND NCCDT

Element	Description
NCCSI	Naming Convention Compliance of Service Interface
NCCR	Naming Convention Compliance of Roles
NCCO	Naming Convention Compliance of Operations
NCCP	Naming Convention Compliance of Parameters
NCCDT	Naming Convention Compliance of Data Types
NCC(me)	Naming Convention Compliance: set of elements out of the set of modelling elements me that follow specified naming conventions

The used names do not correspond to naming conventions specified in the project. For example, spaces are not allowed within names, which is why the NCCSI for the Observation Provider service interface returns 0.

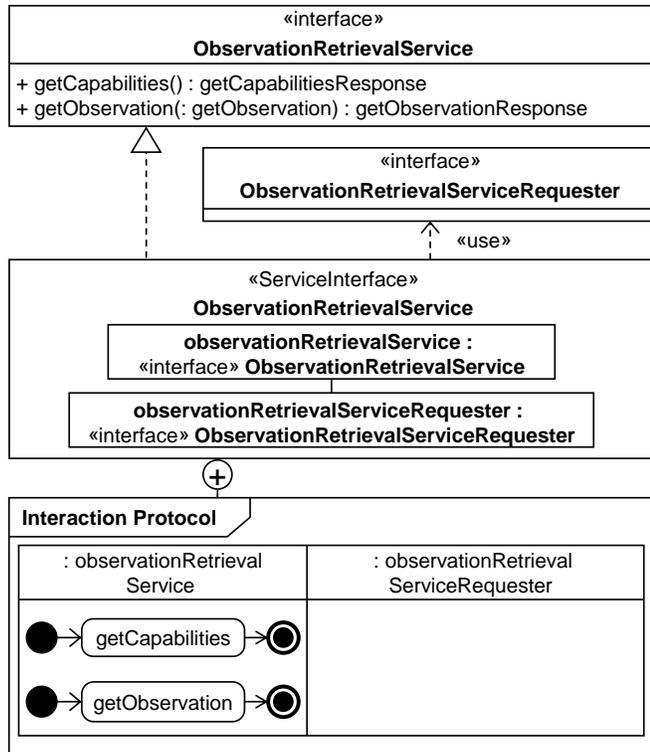


Figure 12. Revised service interface.

TABLE XIX. INTERPRETATION OF VALUES FOR NCCSI, NCCR, NCCO, NCCP, AND NCCDT

Value	Interpretation
Less than 1	There are elements that do not follow naming conventions
1	All elements follow naming conventions

As result, the names of the artifacts have to be revised in order to fulfill the naming conventions and support the discoverability. Figure 12 shows the revised service interface for the Observation Provider.

Whether all possible five information is provided can be measured by the following metric.

$$IC(s) = \frac{EX(SI(s)) + EX(RI(SI(s))) + EX(UI(SI(s))) + EX(R(SI(s))) + EX(IP(SI(s)))}{5}$$

TABLE XX. VARIABLES AND FUNCTIONS USED FOR IC

Element	Description
IC	Information Content
EX(e)	Exists: returns 1 if the element e exists, else 0
IP(si)	Interaction Protocol: interaction protocol of the service interface si
UI(si)	Used Interfaces: used interface provided by the service consumer

As in this article all information is provided, the metric returns 1 for all services.

TABLE XXI. INTERPRETATION OF VALUES FOR IC

Value	Interpretation
Less than 1	Within the service design not all possible information is available
1	All possible information is available

3) *Loose Coupling*: Most quality indicators for loose coupling were not measurable on basis of service candidates. Thus, entirely new metrics have to be introduced.

The asynchrony for long-running operations can be determined as follows.

$$ASYNC(s) = \frac{|ASO(IP(SI(s))) \cap LRO(O(RI(SI(s))))|}{|LRO(O(RI(SI(s))))|}$$

TABLE XXII. VARIABLES AND FUNCTIONS USED FOR ASYNC

Element	Description
ASYNC	Asynchrony
ASO(ip)	Asynchronous Operations: asynchronous operations within the interaction protocol ip
LRO(o)	Long Running Operations: long-running operations out of the set of operations o

Whether an operation is provided synchronously or asynchronously can be determined by means of the “synchronous” flag of a UML CallOperationAction within the interaction protocol. As there is no long-running

operation in this scenario, the metric is not defined and cannot be determined. Otherwise the results can be interpreted as follows.

TABLE XXIII. INTERPRETATION OF VALUES FOR ASYNC

Value	Interpretation
Less than 1	There are long-running operations that are not provided asynchronously
1	All long-running operations are provided asynchronously

The common data type complexity is measured by the following metric.

$$CDTC(s) = \frac{\left| SDT \left(\begin{array}{c} DT \left(P \left(O \left(RI(SI(s)) \right) \right) \cap \\ DT \left(P \left(O \left(RI(SI(ALL_s \setminus s)) \right) \right) \right) \end{array} \right) \right|}{\left| DT \left(P \left(O \left(RI(SI(s)) \right) \right) \right) \right|}$$

TABLE XXIV. VARIABLES AND FUNCTIONS USED FOR CDTC

Element	Description
CDTC	Common Data Types Complexity
SDT(p)	Simple Data Types: simple data types within the parameters pt

The service designs in the considered scenario use own packages for own data types, i.e., they do not have common complex data types. Within the numerator the intersection is empty, which is why the metric returns 0 for all services. As the values 0 or 1 represent desired ones, there is no revision necessary.

TABLE XXV. INTERPRETATION OF VALUES FOR CDTC

Value	Interpretation
0	There are no common data types used
Between 0 and 1	There are common and complex data types used
1	The commonly used data types are simple

The following metrics measure the abstraction of operations and parameters.

$$AO(s) = \frac{\left| A \left(O \left(RI(SI(s)) \right) \right) \right|}{\left| O \left(RI(SI(s)) \right) \right|}$$

$$AP(s) = \frac{\left| A \left(P \left(O \left(RI(SI(s)) \right) \right) \right) \right|}{\left| P \left(O \left(RI(SI(s)) \right) \right) \right|}$$

TABLE XXVI. VARIABLES AND FUNCTIONS USED FOR AO AND AP

Element	Description
AO	Abstraction of Operations
AP	Abstraction of Parameters
A(o)	Abstract: set out of operations o that are abstract
A(p)	Abstract: set out of parameters p that are abstract

As the operations and parameters are derived from business requirements, they are abstract by nature and do not contain any technical details. The metrics return 1 for all services. This again represents the desired value, which is why there is no further revision required.

TABLE XXVII. INTERPRETATION OF VALUES FOR AO AND AP

Value	Interpretation
Less than 0	There exist operations respectively parameters that are not abstract
1	All operations respectively parameters are abstract

Determining the compensation is similar to the one on basis of service candidates.

$$CF(s) = \frac{\left| CFP \left(SC \left(NC \left(O \left(RI(SI(s)) \right) \right) \right) \right) \right|}{\left| SC \left(NC \left(O \left(RI(SI(s)) \right) \right) \right) \right|}$$

As there have been no changes on service designs, the results are the same as on basis of service candidates.

4) *Autonomy*: Instead of using the dependencies between service candidates the required services of a service component can be considered to determine the dependencies to other services.

$$SD(s) = \left| RS(SCT(s)) \right|$$

The values for the metric are the same as on basis of service candidates, i.e., the metric returns 2 for the Observation Provider and 0 for the other services.

The functional overlap is determined by the following metric, which returns 0 for the Observation Provider.

$$FO(s) = \frac{\left| OF \left(O \left(RI(SI(s)) \right), O \left(RI(SI(ALL_s \setminus s)) \right) \right) \right|}{\left| O \left(RI(SI(s)) \right) \right|}$$

In a next step, the analysis and revision phase is iteratively repeated until there is no further revision necessary. This is why the service design phase ends at this step for the considered scenario. As result, the analysis and revision phase enabled to create service designs with verifiable fulfilled quality indicators. This will support common and wide-spread quality attributes and strategic goals, such as a high maintainability, flexibility and cost-efficiency.

V. CONCLUSION AND OUTLOOK

In this article, the creation of a service-oriented system with quality attributes kept in mind was demonstrated by a geographical information system. As these kinds of systems access distributed information and are expected to be accessible from other systems, an architecture with service-oriented design principles is necessary. Since strategic goals are associated with this decision, the services within the system have to follow certain quality attributes, such as loose coupling and autonomy. As concrete scenario a system of the PESCaDO project of the European Commission was chosen.

After an introduction and the definition of relevant terms in the Background section, the scenario and the artifacts of the business analysis phase were presented. The considered business use case described the requirement to get an observation by using different other services to ensure that all relevant information for the user are found and retrieved. The resulting business process served as the input for the second phase in the service development process, the service design phase, which was performed afterwards.

The service design phase consists of the combination of a systematic derivation of artifacts and the subsequent analysis and revision. The first enables the fulfillment of functional requirements and the latter ensures the compliance with non-functional ones, such as the quality attributes. As result, formalized service designs based on SoaML were created for the PESCaDO scenario that consider quality attributes and thus support the achievement of strategic goals.

With this systematic approach, the IT architect is assisted with performing the complex service design task. The application of this approach on a real world scenario exemplifies its usage and shows its benefits. On the one hand, the methodology enables the creation of service designs in an engineering manner. On the other hand, the quality indicators provide a catalog of criteria an IT architect has to consider during this task. This ensures that important quality aspects are not overseen. Additionally, the metrics help with analyzing models and improving them cost-efficiently.

The usage of SoaML as emerging standard for modeling service-oriented architectures and service designs enables the embedding of this approach into existing tools and entire tool chains. As SoaML provides a UML profile, any tool supporting UML can be used. However, there exist also several tools supporting SoaML natively. The possibility to apply this approach with common and wide-spread tools increases its practical applicability.

In the future, we plan to enhance the analysis methodology. There are some terms that are not concretely defined within existing work. For example, when is a service agnostic and when specific? In order to avoid ambiguity these terms have to be specified in detail. Additionally, the quality analysis is supposed to consider further quality attributes especially with regards to the internal component-oriented architecture that implements the service components. Also specifics of paradigms for realizing services, such as the resource-centric approach used in RESTful Web services, will be considered.

Finally, to further increase the cost-efficiency and productivity of the service design task we have implemented the metrics within our QA82 Architecture Analyzer tool [9] for an automatic analysis of service designs. Thus, in the future, IT architects, developers, executive board, or customers will be able to automatically evaluate the quality of developed or acquired products and provided services. This simplifies the analysis whether services increase the flexibility, maintainability, and cost-efficiency of the IT.

REFERENCES

- [1] M. Gebhart, S. Sejdovic, and S. Abeck, "Case study for a quality-oriented service design process", Sixth International Conference on Software Engineering Advances (ICSEA 2011), Barcelona, Spain, October 2011, pp. 92-97.
- [2] W3C, "Web Services Description Language (WSDL)", Version 1.1, 2001.
- [3] OMG, "Service oriented architecture modeling language (SoaML) – specification for the uml profile and metamodel for services (UPMS)", Version 1.0, 2012.
- [4] M. Gebhart and S. Abeck, "Metrics for evaluating service designs based on soaml", International Journal on Advances in Software, 4(1&2), 2011, pp. 61-75.
- [5] M. Perepletchikov, C. Ryan, K. Frampton, and H. Schmidt, "Formalising service-oriented design", Journal of Software, Volume 3, February 2008.
- [6] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-Oriented design", Australian Software Engineering Conference (ASWEC 2007), 2007.
- [7] M. Hirzalla, J. Cleland-Huang, and A. Arsanjani, "A metrics suite for evaluating flexibility and complexity in service oriented architecture", ICSC 2008, 2008.
- [8] S. W. Choi and S. D. Kimi, "A quality model for evaluating reusability of services in soa", 10th IEEE Conference on E-Commerce Technology and the Fifth Conference on Enterprise Computing, E-Commerce and E-Services, 2008.
- [9] Gebhart Quality Analysis (QA) 82, QA82 Architecture Analyzer, <http://www.qa82.de>. [accessed: July 11, 2012]
- [10] The PESCaDO Consortium, "Service-based infrastructure for user-oriented environmental information delivery", EnviroInfo, 2010.
- [11] W. van den Heuvel, O. Zimmermann, F. Leymann, P. Lago, I. Schieferdecker, U. Zdun, and P. Avgeriou, "Software Service Engineering: Tenets and Challenges", 2009.
- [12] M. Gebhart, "Service Identification and Specification with SoaML", in Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments, Vol. I, A. D. Ionita, M. Litoiu, and G. Lewis, Eds. 2012. IGI Global. ISBN 978-1-46662488-7.
- [13] T. Erl, Service-Oriented Architecture – Concepts, Technology, and Design, Pearson Education, 2006. ISBN 0-13-185858-0.
- [14] IBM, "RUP for service-oriented modeling and architecture", IBM Developer Works, http://www.ibm.com/developerworks/rational/downloads/06/rmc_soma/, 2006. [accessed: July 11, 2012]
- [15] U. Wahli, L. Ackerman, A. Di Bari, G. Hodgkinson, A. Kesterton, L. Olson, and B. Portier, "Building soa solutions using the rational sdp", IBM Redbook, 2007.
- [16] A. Arsanjani, "Service-oriented modeling and architecture – how to identify, specify, and realize services for your soa", IBM Developer Works, <http://www.ibm.com/developerworks/library/ws-soa-design1>, 2004. [accessed: July 11, 2012]
- [17] A. Erradi, S. Anand, and N. Kulkarni, "SOAF: An architectural framework for service definition and realization", 2006.
- [18] P. Kroll and P. Kruchten, The Rational Unified Process Made Easy, a Practitioner's Guide to the RUP, Addison-Wesley, 2003.

- [19] M. Gebhart, M. Baumgartner, and S. Abeck, "Supporting service design decisions", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 76-81.
- [20] P. Hoyer, M. Gebhart, I. Pansa, S. Link, A. Dikanski, and S. Abeck, "A model-driven development approach for service-oriented integration scenarios", 2009.
- [21] M. Gebhart and J. Bouras, "Mapping between service designs based on SoaML and web service implementation artifacts", Seventh International Conference on Software Engineering Advances (ICSEA 2012), Lisbon, Portugal, November 2012, pp. 260-266.
- [22] S. Johnston, "UML 2.0 profile for software services", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/05/419_soa/, 2005. [accessed: July 11, 2012]
- [23] OMG, "Unified modeling language (UML), superstructure", Version 2.2, 2009.
- [24] J. Amsden, "Modeling with soaml, the service-oriented architecture modeling language – part 1 – service identification", IBM Developer Works, <http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.html>, 2010. [accessed: July 11, 2012]
- [25] M. Gebhart, M. Baumgartner, S. Oehlert, M. Blersch, and S. Abeck, "Evaluation of service designs based on soaml", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 7-13.
- [26] T. Erl, SOA – Principles of Service Design, Prentice Hall, 2008. ISBN 978-0-13-234482-1.
- [27] M. Gebhart and S. Abeck, "Quality-oriented design of services", International Journal on Advances in Software, 4(1&2), 2011, pp. 144-157.
- [28] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA – Service-Oriented Architecture Best Practices, 2005. ISBN 0-13-146575-9.
- [29] M. Gebhart and S. Abeck, "Rule-based service modeling", The Fourth International Conference on Software Engineering Advances (ICSEA 2009), Porto, Portugal, September 2009, pp. 271-276.
- [30] T. Erl, SOA – Design Patterns, Prentice Hall, 2008. ISBN 978-0-13-613516-6.
- [31] T. Erl, Web Service Contract Design & Versioning for SOA, Prentice Hall, 2008. ISBN 978-0-13-613517-3.
- [32] Fraunhofer Institute of Optronics, System Technologies and Image Exploitation, "D8.3 Specification of the pescado architecture", Version 1.0, 2010.
- [33] S. Johnston, "Rational uml profile for business modeling", IBM Developer Works, <http://www.ibm.com/developerworks/rational/library/5167.html>, 2004. [accessed: July 11, 2012]
- [34] J. Heumann, "Introduction to business modeling using the unified modeling language (UML)", IBM Developer Works, <http://www.ibm.com/developerworks/rational/library/360.html>, 2003. [accessed: July 11, 2012]
- [35] OMG, "Business process model and notation (BPMN)", Version 2.0 Beta 1, 2009.
- [36] W3C, "OWL 2 web ontology language (OWL)", W3C Recommendation, 2009.
- [37] M. Horridge, "A practical guide to building owl ontologies using protégé 4 and co-ode tools", <http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/>, Version 1.2, 2009. [accessed: July 11, 2012]

An Agile Driven Architecture Modernization to a Model-Driven Development Solution

An industrial experience report

Mina Boström Nakićenović
SunGard Front Arena
Stockholm, Sweden
email: mina.bostrom@sungard.com

Abstract–This paper concerns model-driven development (MDD) used in time critical development. We present an agile MDD process developed in consideration of lean and agile development principles and we show its application to the evolutionary development of a real world application supplied to the banking sector. Our approach involves a novel use of concurrent reverse and forward engineering and through our industrial report we are able to provide strong support in favor of the claim that MDD and agile practices can be used together, preserving the benefits of each.

Keywords-agile; lean; MDD; TDD; reengineering; finance

I. INTRODUCTION

In the world of rapid software development, commercial software companies have to respond quickly to the challenges of volatile business environments in order to achieve a fast time-to-market delivery, necessary for surviving on a tough business market [2]. The incoming requirement changes can concern business functionality or technology or both aspects, demanding adjustments and improvements in the existing systems. Adaptations to the frequent business requirements changes can be fulfilled either through the evolution of existing software systems or through the development of new software systems. The direction and quality of the system evolution is steered by three main drivers: system architecture, organizational structure and development process. System evolution often requires adjustments in all mentioned areas. Therefore, the software systems architecture together with the company's organizational structure and the established development process should constantly be adjusted.

Agile software development techniques have been established in order to help organizations both to evolve and to develop software systems, accelerating delivery time while still maintaining, or even improving, product quality [3]. Many companies have started using the agile techniques to a less or larger extent. Important questions that are constantly rising are: how to combine agile techniques with some other, already existing techniques and methodologies? Agile principles present general ideas and recommendations, but they have not been elaborated enough to be specific on

how to work in a particular environment. With the acceptance of agile techniques, the agile principles are also adapting to the different organizations, working environments and methodologies [4]. There are a lot of empirical studies on the agile principles applied on different methodologies, but there is still a need for more empirical results within certain areas. One such area is the application of agile techniques in a Model-Driven Development (MDD) environment. The agile methods and the MDD have appeared separately and evolved on distinct paths, although they address, to a certain extent, the same goals: making systems less sensitive to frequent changes and an accelerated development. Generally speaking, the agile techniques mostly address methodological aspects while the MDD approach is more concerned with architectural issues [5]. Therefore, it became interesting to combine these two approaches in order to get a rapid acceleration of the system development.

This paper, being an extended version of [1], is an industrial experience report that describes an architectural modernization process of an existing system. Despite the fact that the system's architecture is going to be radically improved in the future, there was a need to find an intermediate solution, within a short time-frame, which would both eliminate the existing architectural errors, such as data duplication and system inconsistency, and reshape the system to be less vulnerable to the modifications. Therefore, the existing system was supposed to be transited to MDD, but within a short implementation timeframe as a main requirement. Hence, the main aim of the paper is to answer the following questions:

- How agile and lean principles can help the decision making process when producing a MDD solution within a short time frame?
- How the reengineering process to the MDD solution can be accelerated, fitting the given time frame?

The paper is organized as follows: after the introduction, an overview of the agile and lean techniques is presented in Section II. Section III introduces the Model-Driven Development concept discussing its pros and cons. Section IV describes the problem in details. Section V explains the

architectures of both the present and the long-term solution as well as it introduces reasons for having an intermediate solution, which is separately presented in Section VI. The produced intermediate solution, an Agile MDD approach, is presented in Section VII, while the development process is explained in Section VIII. In Section IX are discussed all benefits of applying agile and lean principles on the MDD. Section X presents the related work. Finally we conclude the paper in Section XI.

II. AGILE AND LEAN TECHNIQUES

Methods of agile software development constitute a set of practices for software development that have been created by experienced practitioners [6]. The main aim of the agile methodologies is to develop qualitative and no cost- effective solutions and deliver them quickly. The core of the agile philosophy is expressed in the agile manifesto, consisting of basic agile principles [3]. The manifesto states that the software development should focus on the following:

- Responding on change over following plan
- Working software over comprehensive documentation
- Individuals and interactions over processes and tools
- Customer collaboration over contract negotiation

If the software development is presupposed on the listed postulates, it can result in fast and inexpensive software that satisfies the customer's needs. Agile practices and recommendations give us answers how to apply the mentioned core values on the development process. The suggested development cycles should be iterative and based on building small parts of the systems, which are tested and integrated constantly. Continuous integration, verification and validation are some of the agile practices that help the organization to check that they are building product in a right way and that the right product is built. The organization should also be arranged to support an efficient, agile development. Agile organizational patterns help in creating a highly effective organization [7]. They concern both the organization of different teams (company management, product management, architects, developers, and test) and the way how people should work within these teams. Some of the most frequently applied organizational agile practices are:

- “Self-selecting teams”: The best architectures, requirements and designs emerge from self-selecting teams.
- “Conway’s Law”: An organization should be compatible with the product architecture and the development should follow the organizational structure.

After one decade of the agile methodologies adoption, empirical studies showed that the best effect is achieved when the agile methodologies are applied on the smaller organizations and projects [6], [8], [9]. Extreme programming (XP), as one of the agile methodologies, is most suitable for single projects, developed and maintained

by a single team [10]. For the larger projects and bigger organizations some other methodologies are more suitable.

A lean software development is an adaptation of principles from lean production and, in particular, the Toyota production system to software development. It is based on the seven principles: eliminate waste, amplify learning, decide as late as possible, deliver as fast as possible, empower the team, build integrity and see the whole. The management decisions should be based on a long-term philosophy, even at the expense of short-term financial goals [11]. The decisions should be made slowly, but implemented rapidly. Work load should be limited and systems should be pulled to avoid overproduction. The lean philosophy is more suitable for bigger organizations and larger projects.

Nowadays practice shows that the best effect is achieved when lean and agile practices are combined together. Although it can seem that some of the agile and lean practices are in contradiction, they are not. At the first sight the agile philosophy could be interpreted as a short-term approach, since it says “do not build for tomorrow”, while lean is more a long-term approach. But these two approaches are not contradictory; on the contrary, they are complementing each other. One of the main lean postulates is “decide as late as possible”, which is another way of prevention for “building for tomorrow”. To conclude, both agile and lean principles could be applied together, but to which extent is decided by the type of organization and the type of the project.

III. MODEL-DRIVEN DEVELOPMENT

Model-Driven Development provides an open, vendor-neutral approach to the challenge of business and technology change [12]. This approach makes, on the system architecture level, a flexible system that can respond quickly on frequent changes both in technology and in business requirements. The main goals of the MDD concept are:

- Simplification and formalization of the various activities and tasks that comprise the software system life cycle, through the raised level of abstraction at which the software is developed and evolved.
- Accelerated development, which is achieved by the centralized architecture and automatic generations.
- Separations of concerns both on technical and business aspects, making the system architecture flexible for the changes.

The MDD's intent is to improve software quality, reduce complexity and improve reuse through the work at the higher levels of abstractions cleared from the unnecessary details. Prominent among the MDD initiatives is OMG's Model-Driven Architecture (MDA) in which software development consists of series of model transformation steps, which starts with a high level specification using often a domain-specific language (DSL), specific for the certain domain, and which ends with a platform-specific models describing how the system should be implemented on certain platforms [13]. MDA standard defines different model categories:

- Computation Independent Model (CIM), representing the problem domain.
- Platform Independent Model (PIM), representing the solution domain without platform specific details.
- Platform Specific Model (PSM), representing the solution domain with platform specific details.

The division could be done even more granular so that the PIM splits in Architecture Independent Model (AIM) and Architecture Specific Model (ASM). Then the PSM is derived from the ASM [14]. All mentioned divisions provide a good separation of concerns. Working with different types of models, representing different views and aspects of the system, enables an easier understanding of complex systems. MDD is intended for the realm of large, distributed industrial software development and is one approach for solving the software life-cycle development problem. On the other side, the detailed separation of concerns can introduce some other problems in the system. It can cause an additional complexity, requiring the existence of several models describing the same thing, just on the different abstraction levels or from the different point of views. Therefore, it is questionably if the MDD reduces the complexity or it just moves the complexity elsewhere in the development process [15].

Development processes based on the MDA are not widely used today because they are considered as heavy-weight processes, which cannot deliver small pieces of software incrementally [16]. That is why there is a need to rework a MDA to a lighter process, easier to the acceptance. For example, this could be achieved by the introduction of agility in the MDD philosophy.

A. Agility in Model-Driven Development.

A common goal for the MDD and the agile methodologies is to build systems, which can respond quickly on the frequent changes. These two methodologies have different approaches for resolving the mentioned requirement: agile development concentrates on individual software products, while MDD is concerned with product lines, i.e., mass-produced software. Agility mostly addresses methodological aspects while the MDD approach is more concerned with architectural issues [5].

The MDD concept has some drawbacks, which do not suit agile philosophy. Looking from the agile perspective, systems should be built in an incremental way where the small pieces of software are delivered constantly. In contradiction to the MDA modeling's starting curve, which can take a long time before the deliverables are produced. True domain-specific languages are not very agile because they encode commonalities and variations in a narrow, concrete expression of the business form [17]. DSL makes the system being too specific decreasing a possibility to respond to the business changes quickly. If the domain evolves, then the language must evolve with it, otherwise the previously written code becomes obsolete. MDA systems usually become complex while agile claims that "simplicity is essential". People are not an explicit feature in MDD while

agile postulates that people and interactions should be over process and tools.

[18] distinguished generative MDD and agile MDD. Generative MDD, epitomized OMG's MDA, is based on the idea that people use very sophisticated modeling tools to create a very sophisticated models that they can automatically transform with those tools to reflect the realities of various deployment platform. [19] proposes the agile MDD, where the agile modeling is used. Agile modeling is practices-based and consists of collection of values, principles and practices. Agile models are models that are barely good enough, where the fundamental challenge with "just barely good enough" is that it is situational and therefore, the most efficient.

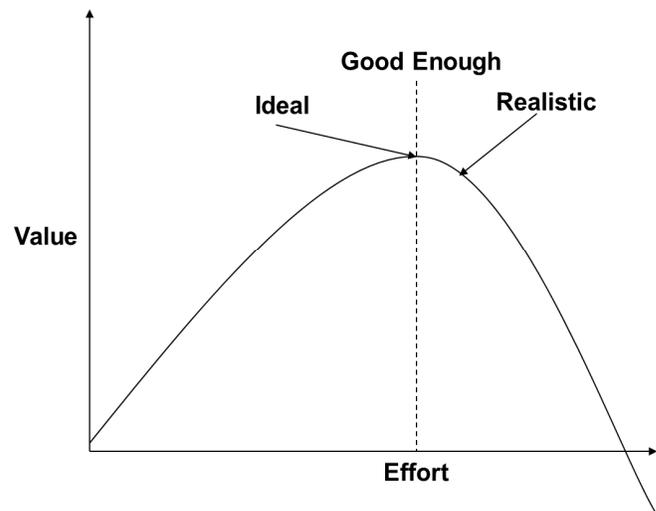


Figure 1 Agile modeling. Adapted from [19]

The main idea with the agile modeling is not to follow strictly the MDA recommendations regarding tools and development environment but to choose ones which fit best the current project and the organizational structure.

IV. PROBLEM DESCRIPTION

A. Background

SunGard is a large, global financial services software company. The company provides software and processing solutions for financial services. It serves more than 25000 customers in more than 70 countries. SunGard Financial Systems provides mission-critical software and IT services to institutions in virtually every segment of the financial services industry. We offer solutions for banks, capital markets, corporations, trading, investment banking, etc. [20]. In several areas SunGard is one of the leading providers for the financial solutions and products. Since the finance industry is very tough, staying on top of the competitive financial market requires fast delivery, reduction of costs and quick responding to the changes in dynamic market conditions. In order to achieve this, our company has started

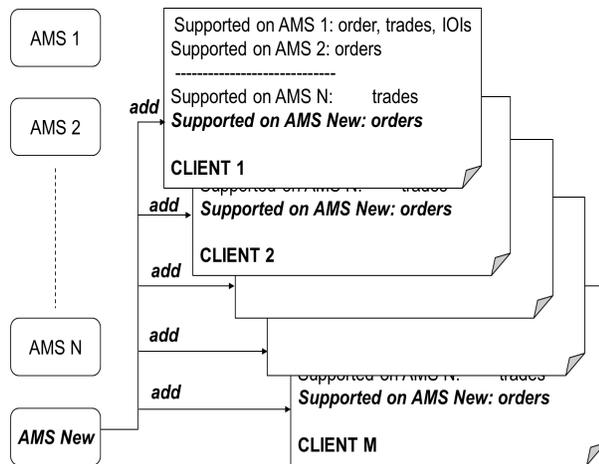


Figure 3. Process flow

The MSCs describe market trading transactions (orders, deals, etc.), the commands that are supported for them (entering, modifying, etc.) and the attributes and the fields, which could be accessed on the markets (quantity, broker, etc.). Hence, specifying the MSC for a new AMS requires a detailed description. All components, which use the MSC functionality, must use the same MSC definition. Unfortunately the same MSCs are defined in several different files. Different components are developed in different programming languages so they do not share the same definition file. Because of historical reasons and the fact that some client components were developed within separate teams, even the components developed in the same programming language do not share the same definition file. Each client component has its own MSC definition file. There is a lot of the duplication of information in these files. Even worse they do not present exactly same data since the different clients work within different business domains, so their knowledge about the MSCs is on the different levels. The described situation arose from bad communication between the teams. Without interacting with each other and without having enough knowledge about the design and the architecture applied on the different projects, it was easy to end up with the described MSC architecture.

B. The present architecture analysis

The client components use the MSC definition from the different sources, developed in different programming languages (C++, C# and Java), where the majority of data is duplicated. This situation, with the usage of the overlapping MSC definitions, is presented on Figure 4.

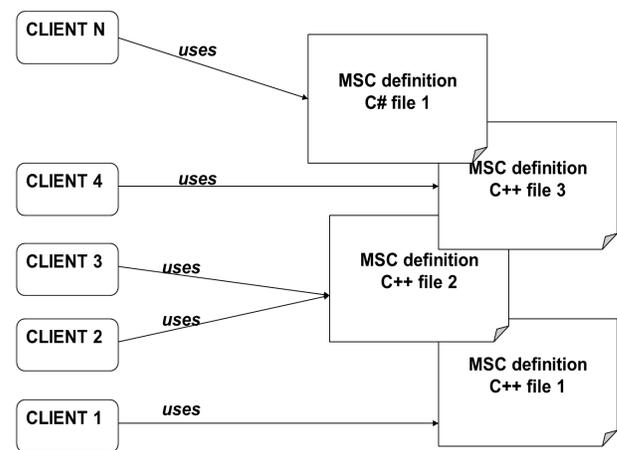


Figure 4. The present architecture with distributed definitions

The present architecture of the MSC definition is not centralized (no single definition of the model) and without control for the consistency. The lack of centralization enormously increases the risk for data inconsistency since the consistency depended on the accuracy of the developers who edits the MSC definition in a source code file. The development of the MSC definition is a continuous process, and new MSCs are defined each time when a new AMS is developed (2-3 times per year) or when a new trading transaction is introduced (once per month). The current process flow is:

- A new AMS is developed or a new transaction is introduced.
- A MSC is added to the MSC definition in each client component. The same information must be added to several different files.
- All client components should be recompiled in order to get the definition of the new MSC.

After the presented files analysis we could state the following facts about them:

- Similar structure: files are structured in the similar way, containing a lot of switch/case statements.
- Data duplication: some data are duplicated
- Different business domains: different levels of the describing aspects are used in the different files.
- Mainly syntax differences: the mainly difference among the files lies in the syntax not in the data structure.

C. Dynamic Market Capabilities architecture

We have already done design plans for the new DMC architecture. In the DMC architecture each AMS will be responsible to provide, to the client components, information about the MSC that the AMS supports. The description of the MSC that the AMS supports will be saved in one XML file.

On the AMS start up, AMS reads the MSC definition from its XML file and sends them, in run time, to all client components, which connect to the AMS. In such a way the client components do not have to be recompiled if something changes in the MSC definition. When a new AMS is developed, a new XML file containing MSC definitions for the AMS is created. On the AMS start up, all client components connect to the AMS and dynamically retrieve the MSC definition for that AMS. In the future, even in this case there will be no need for the recompilation of the client components.

D. Transition phase

The decision is that all AMS components and all client components should be upgraded to the DMC architecture. But this transition is a complicated job. There are over 30 AMS components and more than 5 client components that are using MSC functionality today. There is different prioritizing, from the management side, within the components' backlogs. We know, right now, that some of these components will be upgraded to the DMC in one or two years. This transition project is not marked as a critical since there is already a working architecture, although not the best one. As long as there is at least one component, which has not been upgraded to the new DMC architecture, the hard-coded MSC solution must still be supported. The transition will occur gradually and the transition phase will probably take several years. Under the transition phase some new components are going to be developed; some new components are already under the development. To develop new client components according to the present architecture will introduce even more duplication. Therefore, an intermediate architecture, which will eliminate the duplication, would be introduced. Such a solution should have a short implementation phase, since it must be ready before the new components are completely developed. The solution should be designed so that it eventually leads towards the new DMC architecture. It would be good if the new DMC architecture could benefit from it.

VI. INTERMEDIATE SOLUTION

We work according the lean and agile software development philosophy. One of the key principles of the lean philosophy is to detect and eliminate wastes [22]. The intermediate solution should eliminate, from the present architecture, the three major points of waste.

- Duplication of the MSC information
- Amount of work done during the MSC definition updates
- Amount of time used for communication among groups, informing each other about the MSC definition changes

A. Technical Aspects

The waste elimination adds an important business value, according to the lean philosophy. Even if the transition of the existing MSC architecture to the intermediate architecture

does not directly add a business value from the customer's perspective, the existing system's wastes would be eliminated by the reengineering process. In that way the delivery of the new solutions, which are dependent on the MSC architecture, would be accelerated. Hence, we get an implicit business value, which would be produced by the intermediate solution.

In order to eliminate the duplication of data we needed a centralized MSC definition. In order to be able to provide support for the MSC definition in different programming languages we needed to generate code in different programming languages, from the centralized MSC definition. We need a programming language independent architecture. Because of the lack of time, we decided to have an agile approach on brainstorming meetings when we were searching for the architecture of the intermediate solution. We did not want to waste a time on investigating all possible solutions, since the time was more precious for us than perfection. We suggested and analyzed three different approaches and chose one among them, which was the most suitable. Although we did not analyze all possible solutions, we got a methodology that was good enough. To use a good enough solution for the current situation, within a short time frame, suits the agile philosophy.

First we considered a solution, where all client components would be refactored to reference the same central definition file. This would require a lot of work. We did not want to refactor client's components too often, since some of them will be refactored soon regarding the DMC solution.

A generative programming concept [23], using a parameterized C++ templates, was discussed as the second solution. Such solution would consist of the generated classes, representing the TNP objects (TNPMessages and TNPRecords). The main intention of the generative programming is to build reusable components. A cost of building the reusable components should be paid off by reusing them in many systems. When a goal is to build just one system and when schedule, to deliver a system, is tight, the introduction of the generative programming idea cannot be the best solution. Additionally, the existence of C# MSC definition file made the usage of the C++ templates impossible.

Finally we analyzed the Model-Driven Architecture (MDA) approach. With the MDA approach we mean the general MDA concept: "A MDA defines an approach to modeling that separates the specification of system functionality from the implementation on a specific technology platform". The common denominator for all MDA approaches is that there is always a model (or models), as the central architectural input point, from which different artifacts are generated and developed. Transformations, mapping rules and code generators are called in common "MDA tools" [24].

We believed that the Model-Driven Architecture (MDA) approach would be the most suitable solution for the intermediate architecture. The main idea was to have just one source, a union of all present MSC definition that is

programming language independent. From such a source, which would be a central MSC definition registry, the present MSC definition source files are generated. All present MSC definition files have a similar structure. The main difference is the programming languages syntax. Because of that the code generation should not be too complicated. The way how the client components work would not be changed, the MSC definition would still be hard coded. Such a solution did not require the refactoring of the client components. But the way how the developers work would be improved. They will work just with the central MSC definition registry and add/edit the MSC definition only there. Then the MSC definition files, for each client component, will be automatically generated from the central registry. The client components will be automatically recompiled. In that way all three mentioned wastes will be eliminated.

Another key lean principle is to focus on long-term results, which is the DMC architecture in our case. That is why we must point out that one important part of the DMC architecture is a MSC XML description file. If the MDA approach is introduced for the MSC definition, the central MSC definition registry would be easily divided into several files (one per AMS), later on. It is clear that the DMC architecture would benefit from having such a central MSC registry. The creation of one central MSC definition registry, with all MSC definitions for all markets, would be a good step towards the future DMC architecture introduction.

B. Organizational Maturity and Limitations

Our company management is usually very careful with introducing concepts not already used in the company, since it often requires long implementation and learning time. Additionally, an investment in an intermediate solution is not always a very productive investment. On the other side, the management was aware that the intermediate architecture would increase productivity directly and make some new solutions possible right away. That is why the management listened carefully to our needs and made some general decisions. The intermediate architecture can be introduced, but the time-frame could be only several weeks. No new tools or licenses should be bought. Only tools that are already used within the company or some new, open-source tools, can be used. No investment in change management. Time for teaching/learning cannot be invested for the intermediate solution. The concepts, which our developers are already familiar with, should be used.

Considering these management decisions, we decided to explore if the organization was mature enough to introduce the MDA. Although the MDA approach has been around for a long time, for many companies it is still a new approach. That is why we performed a small survey, with questions presented in Figure 5.

1. *Have you worked in the MDD environment before?*
 - a. *If yes, what is your experience regarding the MDD?*
 - b. *If yes, have used the UML modeling?*
 - b. *If no, have you heard about it?*
2. *Do you think that the MDD approach should be introduced in our project?*

Figure 5. Survey Questions

We asked 60 developers, working in the 6 different teams. 4 teams consisted of C++ developers, 1 team consisted of Java developers and 1 team contained C# developers. The survey showed that the MDA approach hasn't been used within the company and that a majority (80%) of the developers has never used this approach. Consequently the UML modeling is not used in general. Some teams were using MagicDraw, but just as a documentation tool for the state-machines drawing. The architects, who designed the state machines, answered that it was faster to develop own generators, using the state diagrams created in the MagicDraw than to investigate how to use the UML tools and profiles and code generators.

Additionally, there was a previous attempt of introducing the MDD in the enterprise architecture, which unfortunately failed. The former MDD project consisted of a new modeling framework, based on the Eclipse framework, particularly designed for the drawing Front Arena state-machines. The project has never been finished because it took a long time without showing the results. Unfortunately it happened at a bad point of time, when the financial market was extremely poor and when the product delivery to the customer was a matter of the utmost importance. Consequently the company lost time and money by investing in this MDD framework. The main problem was not the MDD concept by itself, but it was difficult to see an explicit business value in it. A time-consuming and cost-effective MDD introduction was in contradiction with a fast and frequent delivery. Because of all mentioned reasons the majority of the developers, as well as the management, did not believe in a new attempt of working with a MDD idea. The introduction of the full scale MDA usually implies: a long starting curve, which we could not afford having a short time-frame and the usage of the MDA tools, which could not be used since developers did not have enough knowledge about them and there was no possibility to invest in learning. In the following section it will be described how we managed to overcome these problems and limitations.

VII. AGILE MDD APPROACH

Our goal was to find an intermediate solution with a MDA philosophy, which satisfied the previously mentioned requirements and fulfills the constraints. In order to achieve this goal, we started from the basics of the MDA concept (models, transformations and code generators), and combined them with the following lean and agile principles [3]:

- "Think big, act small": Think about the DMC as a final architecture but act stepwise, introduce the intermediate solution first.
- "Refactoring": A change made to the structure of software to make it easier to understand and cheaper to modify without changing its existing behavior [25]"
- "Simplicity is essential": We have to find an applicable solution that is simple, keeping in mind that simple does not have to mean simplistic [17].
- "Individuals and interactions over processes and tools": It is important is to find a solution, which fits the developers, as well as to establish such development process, which will be effective in our company.

We used the agile and lean ideas both in the decision making and in the development process. In that way we got our own Agile MDD approach, an applicable intermediate solution.

It is important to emphasize that we had an existing architecture, which should be transformed to the MDD solution. The process of system transformation is called a system reengineering. A system reengineering phenomena has been present in the software development as long as the software systems exist. With the high dynamic of business requirement and technology changes, the systems have to be modernized constantly. System modernization is a way of system adaptation to the changes. Architecture-Driven Modernization (ADM) is an OMG standard for the system modernization [26] and can be briefly described by using the ADM horseshoe model, presented on the Figure 6.

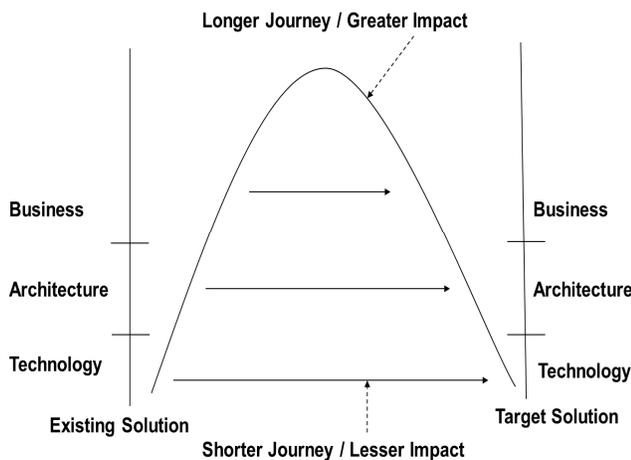


Figure 6. ADM horseshoe model. Adapted from [26].

According to the ADM standard, three areas could be distinguished during the reengineering process; technical, architectural and business area. Depending on the extent of areas that are affected during the reengineering process, the ADM journey can be longer or shorter. Consequently the impact of system changes can be greater or lesser. The duration of the reengineering process directly affects time-to-

market. This fact was important to bear in mind when making the architectural decision within the chosen MDD solution, as it is described in the following sections.

B. Agile modeling

We needed to model the MSC definition registry. This modeling can be done on the different modeling levels and in the different modeling languages. The UML is one the most frequently used modeling language and it became a modeling notations standard, according to the OMG's recommendations. The UML has been developed and evolved to cover many different needs, becoming, at the same time, huge and unwieldy. Although the UML profiles have been introduced in order to help the developers to exclude unneeded UML parts, there are still many cases where the adoption of the MDD has been slowed because of the UML's complexity [15]. Additionally the UML lacks sufficient precision to enable complete code generation [27]. The time frame for our project was short and the developers were without enough UML experience, since the UML is not used in general. According to the limitations, there was no time for learning. Hence, the UML modeling could not be accepted as a modeling solution in our project. Since the XML format is a standard format and the developers are familiar with it, we decided to use a XML description as a "natural language" for the developers. XML was good enough. We had to balance between the familiarity of the XML and abstraction benefits of UML but also a complexity of the related frameworks, keeping the project within the time-frame. Also the XML usage would imply the shorter journey for the reengineering process, compared to the long journey required for the reengineering to the UML model.

The MDA defines different model categories, like a Platform Independent Model (PIM) and a Platform Specific Model (PSM) [24]. As discussed before, although the multi-model concept provides a good separation of concerns, at the same time it could introduce an unnecessary waste in the system, which is in contradiction with a lean architecture. Hence, the multi model concept should be used only if there is a really need for that and when a separation of concerns is required in order to be able to understand and work with a system. The PIM and PSM concept becomes an important issue if there are plenty of different platforms with specifications that differ very much. In our case the different PSMs did not differ too much from each other and, at the same time, did not differ too much from the PIM either. In order to keep it simple we made a pragmatic solution: to have just one model, which contained all info for all programming languages. The code generators had the responsibility for creating the right MSC information to the corresponding programming language.

```

<MarketCapabilities>
<MarketCapability id="200001" HomeMarketServerId="OMX" version="1.0.0" >
<Objects>
<Object name="QuoteRequest" value="RTY_QUOTEREQUEST">
<Commands>
<Command name="EnterQuoteRequest" value="RTY_ENTERQUOTEREQUEST"/>
<Command name="RejectQuoteRequest" value="RTY_REJECTQUOTEREQUEST"/>
</Commands>
<Fields>
<Field name="BidOrAsk" value="RTY_BIDORASK enter="yes"
EnumId="MyQuoteRequestBidOrAsk"/>
<Field name="Quantity" value="RTY_QUANTITY" enter="yes"/>
</Fields>
</Object>
<Object name="Order" value="RTY_ORDER">
<Commands>
<Command name="EnterOrder" value="RTY_ENTERORDER"/>
<Command name="ModifyOrder" value="RTY_MODIFYORDER"/>
<Command name="DeleteOrder" value="RTY_DELETEORDER"/>
</Commands>
<Fields>
<Field name="QuantityCondition" value="RTY_QUANTITYCONDITION"
enter="yes" EnumId="MyQuantityCondition"/>
<Field name="PriceCondition" value="RTY_PRICECONDITION"
enter="yes" modify="yes" EnumId="MyPriceCondition"/>
<Field name="MarketAccount" value="RTY_MARKETACCOUNT"
enter="yes" modify="yes" EnumId="MyMarketAccounts"/>
<Field name="OrderAttributesHidden" value="RTY_ORDERATTRIBUTES"
enter="yes"/>
<Field name="AMASFixOrderState" value="RTY_AMASFLXORDERSTATE"/>
</Fields>
</Object>
<Object name="PriceDetail" value="RTY_PRICEDETAIL">
<Commands>
<Fields>
<Field name="LastPrice" value="RTY_LASTPRICE"/>
<Field name="LastQuantity" value="RTY_LASTQUANTITY"/>
<Field name="LastDate Time" value="RTY_LASTDATETIME"/>
<Field name="HighPrice" value="RTY_HIGHPRICE"/>
<Field name="LowPrice" value="RTY_LOWPRICE"/>
<Field name="ClosingPrice" value="RTY_CLOSINGPRICE"/>
</Fields>
</Object>
...
</Objects>
</MarketCapability>
...
</MarketCapabilities>

```

Figure 7. XML Model

We have created two models. One was a logical model that describes the entities in the MSC definition registry. Another was the MSC definition registry by itself, expressed in a XML dialect, which is presented on Figure 7. Consequently the logical model was expressed as a XSD schema and was used to validate the entries in the registry.

C. Code generators

We needed code generators for generating the different types of files: C++, C#, Java. We decided to use XSL transformations as the code generators. They satisfied our needs and could be widely used, since the XSL is a common standard for all developers, who program in the different programming languages. In that way a "collective code ownership" [3] is achieved for the code generators. The maintainability is also better if all developers can maintain/develop the transformations.

VIII. THE DEVELOPMENT PROCESS

Our company has introduced the agile software development several years ago. Scrum is used as a process tool. Each team runs its own sprints, typically lasting for 4-5 weeks. The sprints are synchronized, meaning that the start and the end sprint date is the same for all sprints within the company. Such synchronization makes the releases and the

delivery process of the dependent components easier. Although we use Scrum, all teams do not strictly follow all Scrum recommendations, it is more up to the team how the Scrum is performed, dependent on the currently running project.

The intermediate solution, as an internal project, was supposed to be done in parallel with other running projects. In order to fit in the company's culture, we decided to run our project according to Scrum, based on the sprints. In general, working in Scrum sprints suited our project well. When we worked on the common data, which were present in several MSC definition files, it was easy to plan the coming sprints, since we knew the next required steps. For example, we knew that we should extract all capabilities per market. Scrum suited well for the major parts of the project as we were planning one sprint at time.

On the end of project, when only odd data, specific for a certain market or a certain component, was left it was difficult to plan the sprints. When we had many small tasks, which were not very related to each other and which were not easy to separate and divide into the sprint tasks, Kanban [11] was more suitable. Therefore, when we were approaching the end of the project, we switched our development process to a Kanban. In contrast to Scrum, tasks in Kanban are performed one after the other, without collecting them into sprints. One of the main Kanban principles is to limit "work in progress" by defining the maximum of tasks, which can be performed in parallel. If this number is exceeded, no new tasks are taken from the backlog until there is an available capacity for a new task. Changes to the product backlog take effect as soon as capacity becomes available. A typical Kanban board is presented on the Figure 8 where both short and long running tasks can be executed in parallel.

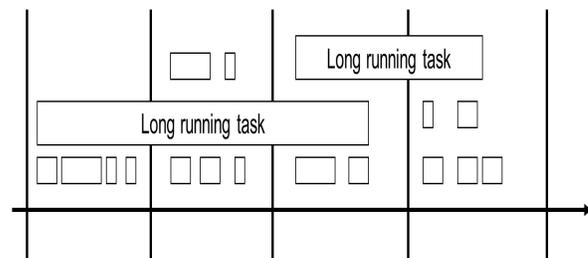


Figure 8. Our Kanban process. Adapted from [11].

Since we could not appreciate time for the tasks that were left, we just put them on the board and took them as soon as the previous task was finished.

A. Team Selection

A good team communication is one of the necessary prerequisite for a successful development [2]. The absence, irregularity and incompleteness in communication among the company's teams caused the duplication and inconsistency in the present MSC architecture. According to the agile

manifesto people and interactions should be over tool and processes [3]. As the reengineering to the MDD solution was a new challenge for the developers in our company, it was important to have a “self-selected” team [3], with the developers that were interested and willing to work on it. Since the main part of our project consisted of working with the legacy code, we wanted to have experts in the team, with deep domain knowledge about all existent MSC files. Applying organizational patterns “architects also implements” [7], three architects from different teams were chosen, one for each MSC product owner team. In that way we got a good expertise for different business domains, for all types of the MSC files. On the other side, the chosen experts worked together in a pair-development sessions, supporting the concept of “generalizing specialists”. Generalizing specialists are often referred to as craftspeople, multi-disciplinary developers, cross-functional developers and deep generalist [28]. It was important that the experts working on the project could see the whole impact of the changes, not only within their expertise domain.

Another important task, related to the team communication, was informing all teams, which were the product owners of the MSC files, about this project. We wanted to avoid making the same mistake as it was done before. Therefore, it became very important with sprint demos, combined with the result presentation, for all affected teams. The two important purposes of demos were:

- Spread the knowledge about the done and to-be done project tasks
- Show that the MDD project can be rapidly developed.

Additionally, the knowledge spread was even more effective with the chosen experts, belonging to the different teams, since each expert talked to its colleagues about the ongoing project.

B. Reverse engineering of the Legacy code

We needed to do a one-time reverse engineering in order to convert a large amount of the existing MSC data, legacy code, to the new MSC XML format. We developed our own tool for this purposes since no open-source tool was completely suitable. The main question was: when to start with the reverse engineering? At the end or at the beginning of the project? Very soon we realized that we could not design our model in detail without the data from the existing MSC definitions. It was data stored in the MSC definition, which lead the reengineering process. This data became a kind of business requirement in our project. Consequently the requirements were not developed; they were discovered during the reversing process.

We decided to adopt a spike principle. The spike is a full cross-section of the modeling and architecture aspects of the project for a specific scenario. The aim of the spike approach is to develop the whole chain for only one, chosen user scenario. The first chosen scenario is a simple one, and during the incremental development process every next scenario is a more complex one [29]. We started with the

round-tripping (the whole chain: model – code generation – reversing back to the model) for simple scenarios, which we expanded, in each sprint, to the more complex scenarios. In that way we could develop the reverse engineering tool, the code generators and to design the model in parallel. The results of the reverse engineering helped us with the specification of the model objects for both the logical model and for the central MCS registry. Working in that way, we allowed “the business requirements coming late in the project”. In our case, the business requirements were mainly the results (predictable and unpredictable) from the reversing process, which steered the reengineering project. Since we could do the round-tripping very early in the project, it was a way in which we could start testing our MDD approach early, under the development.

C. Round tripping with the TDD approach

According to the lean principles, we wanted to specify our model just according to the existing data, without unnecessary objects or unnecessary properties, which risk never to be used. In order to be able to do that, we wanted to do the reversing first and specify the logical model and fill the data in the MSC registry upon these results. We used a TDD approach and started with writing unit tests first. For this purpose we used test framework developed and already used in the company. This framework simulates the execution of the TNP messages sent among server and client components. Because of that the test scenarios that we wrote can be reused later on, for testing AMS components, when the DMC is introduced.

According to the TDD principles we wrote the tests first, run them on “empty” code and developed the code, until the tests passed. Since we had to test several parts of our MDD approach (the logical model, the central MSC registry, the code generators and the reverse engineering tool), we established our own TDD process for the MDD testing. The main idea was to use the same tests, which reflects the parts of one spike, both to develop the reverse engineering tool and the code generators. Our TDD process is presented on the Figure 9 and will be described now through one real spike. The chosen spike is called “Get all markets” and the goal is to get all existing markets, described in the present MSC files. We started with writing a test, which consisted of sending a TNP message “TNPGetAllMarkets”. The next step was to develop the reverse engineering tool for this scenario. The legacy code was used as input data. We developed the corresponding methods in the reversing tool, which extracted markets from the existing data, producing the results in the XML format, and inserted them in our MSC registry. It was a list of all markets. Then we redesigned the model and registry entities and refactored the reversing tool according to the model changes. This process flow is presented with semi-dashed arrows on the Figure 9.

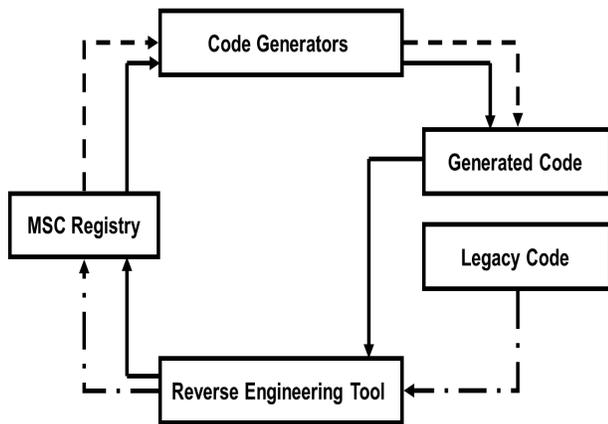


Figure 9. Our TDD process

The TDD logic for the code generators was the following. What we had, so far, was the reversing tool working for the chosen scenario, and some data in the central MSC registry. We used the same test against the code generators, trying to get all markets from the MSC registry. We developed the code generators using the mentioned test. This process is marked with dashed arrows on the Figure 9. The final goal was to get the same code lines in the generated code files, as we had in the corresponding legacy code files, concerning the data affected by the chosen spike. In order to verify this, we run the whole round-trip but this time we used the generated code files as the input for the reverse engineering tool. Eventually we compared the newly generated files with the previous ones and the legacy code files and if there were some differences we adjusted the reversing tool and the code generators accordingly. This process is marked with full arrows on the Figure 9. After this sprint we had a list of all markets in the MSC registry, the code generators methods, which generate files containing such a list, and the reversing tool methods for extracting such a list from the generated files. In the following sprints we used more advanced scenarios, such as, for example, “Get all markets where is Order supported with commands: Enter, Modify”.

At the end of each sprint we run the whole round tripping, starting from the legacy code. In that way we could confirm that both the newly implemented code worked, as well as that the previously implemented code was not broken. As the final verification process we confirmed that all client components could be compiled without errors. We did the usual integration tests also, in order to confirm that the communication among the client components and the AMS components has not been changed. When we completely finished with the reversing, we disabled this functionality. We needed the reversing only for extracting the existing data. It has not been possible do the reversing nor the round tripping since the project was released.

D. Test-first Tests

Two reasons were crucial for choosing the TDD approach in the reengineering project. As first, we believed that the TDD approach could accelerate the development. The second reason was the fact that we did not have enough knowledge about the MSC files content so we did not have a clear idea how to start the implementation. Therefore, writing the tests first was a good start. We usually started with file investigation and wrote the test-first tests as soon as we understood the existing code. It was an excellent start to begin the implementation of one spike. We applied often the “learn it” TDD pattern [30] in order to examine the files and write the test-first tests accordingly.

The introduction of the TDD approach was important because of the following reasons:

- By developing and testing in parallel we shortened the implementation phase.
- We did not produce any wastes in the logical model (unnecessary info). We designed the model just according to the data that we got from the reverse engineering. We achieved to avoid the usual modeling mistake when a large amount of metadata is put in the model.
- The reengineering process was accelerated since the reverse and forward engineering were performed simultaneously.
- We showed how the TDD can be an efficient way to work with, since this development method has not been yet widely spread within the company. When it has been introduced once, it would be easier to introduce the TDD thinking in other projects too.
- We learned a lot about the different TDD patterns.
- We can reuse some of these tests later on, for the DMC architecture testing.

It is important to say that we had to reverse the legacy code from the code, which was written in the different programming languages. We had to develop separate methods for the reversing from C++, Java and C#. Fortunately, the respective legacy code files had a similar structure; the syntax was the main difference. So we could develop the corresponding reversing methods based on the common objects.

E. Continuous Integration with Automation

The continuous integration is a software development practice where the software is integrated frequently, having the integrations verified by automatic builds to detect integration errors [31]. TDD and Continuous Integration (CI) are agile practices, which complete each other. TDD produces code that is well designed and relatively easy to integrate with other code. The incremental addition of small parts to the system, together with the automatic builds, provides the continual system development without extensive integration work [32]. The general continuous integration concept was already introduced in the company. All client and server components, which use the MSC

definition, have the automatic builds enabled. When the code changes in the code repository, which is Clear Case in our case, the automatic builds are started by a trigger scripts. The trigger scripts, integrated in the Clear Case and specially developed for this purposes, are responsible to start the automatic builds on the build servers. If the builds fail, the responsible product owners are immediately notified by email.

On the end of the project, we have automated some of the processes reducing the amount of work and time spent on working with the MSC definition architecture. We use ClearCase (CC) as a configuration management tool and we have a build server for automatic build processes. Since all client MSC definition files were in CC, we decided to keep even the generated files in the CC repository, at least under some period. This decision was made by the management.

When the MSC definition registry file is updated and checked into CC, the following steps are executed automatically:

- The MSC definition files with hard-coded data, belonging to the client components, are checked out from CC.
- The code generators are invoked by a CC trigger script. All MSC definition files are generated.
- All generated files are checked into CC, if the generation did not fail. Otherwise the “undo checkout” operation is done.
- All client components, affected by the mentioned code generation, are recompiled. If some compilation fails, the error report is immediately sent to the component owners.

Continuous integration verified that all parts of the MDD solution are synchronized. To clarify, previously existing tests are run against the generated files, as they were run against the hard-coded files before. In that way we had an automated check that the legacy code was not broken. New test suit, containing tests used for the code generators development, were also added to the automated test execution. The corresponding tests were not run against the reverse engineering tool, since this functionality was disabled on the end of the project.

F. Light-weight Documentation

Although the documentation generation was not among the project requirements, the MDD solution enabled a possibility to generate documentation about the MSC for the different markets in a light way. The MSC registry, expressed in the XML format, supported a possibility for writing comment lines. In that way we could easily develop a generator that generates a HTML files presenting different MSC aspects. For example, beside basic tables describing the supported capabilities on the separate markets, it was interesting to create lists for each capability, presenting all markets where the capability is supported. The latter information was very useful for the product management team, to get quick information about the market capabilities. The user-friendly presentation was highly appreciated, since

it shortened time when searching for information. In the described way we achieved to get light-weight documentation, which is easy to update and does not cost much to maintain. Thanks to the XML format of the MSC registry, the documentation generators development was a trivial job, lasting for just one developer day. Such way of documenting MSC definitions fitted well the agile philosophy.

G. Results

The project was completed within the 4 sprints, lasting for 4 weeks each, and one month of Kanban process. At the early stage of the project, without enough experience, we could not plan the first sprint in the most efficient way. It was during the first sprint, which took more than the one month, when we made the decision to do the reverse and the forward engineering in parallel. After that, the development was accelerated as well as the sprint’s velocities. Velocity of the first sprint was only 10 story points. Every next sprint was executed with a velocity of 15 to 20 story points. We planned the coming sprints according to the results and experience from the previous sprints. During the Kanban process, the development speed decreased again since we were stacked with a lot of small problems which were supposed to be solved separately. On the other side, the fact that we were approaching the end of the project encouraged us with completing the tasks. Although we could have completed the project several weeks earlier, if we planned the first sprint better, the management was satisfied with the performed results. The extenuating circumstance was the fact that the intermediate solution was an internal project, without fixed released date to the customer.

IX. AGILE AND LEAN PRACTICES IN MDD

The agile and lean methods are light in contrast to the MDD that can become complex. Through the application of the agile and lean principles, the MDD becomes more pragmatic and more useful. Some of the agile and lean principles, used in our Agile MDD approach, are explained below.

A. Architectural aspects

“Eliminating waste” Eliminating the duplication of information was also according to the XP’s principle “Never duplicate your code” [33]. This principle is the heart of the MDD – to have one central input point, model (models) from which everything else is generated.

“Think big, act small” We were thinking on the DMC as a final architecture but acted in a stepwise way, via an intermediate solution.

“Simplicity is essential.” We have simplified the full scale MDA. Instead of the UML modeling language we used the XML. The PIM and PSMs were merged, avoiding the maintenance of several models and transformations among them. On the other side, by merging PIM and PSMs in one model we lost a good separation of concerns but it was a price worth paying.

B. Organizational aspects

“Self-organizing teams” contributed to the successful MDD introduction, since the evolved people were indeed interested to complete the project.

“Empower the team” Roles are turned – the managers are taught how to listen to the developers [22]. Despite the fact that the management puts non-technical constraints on our project, they allowed the developers to make decisions, regarding the intermediate solution, on their own. It contributed to faster development, since the developers did not have to wait for feedback from the management, for each decision.

“We became a constantly learning organization, through relentless reflection and continuous improvement.” Since the organization was without enough previous knowledge within the MDD area, we learned a lot about applying this concept in practice.

C. Development process aspects

“Deliver as fast as possible”. The implementation phase of our Agile MDD approach was short.

“Spike principle” applied on the round-tripping, which includes both the reverse and the forward engineering, made the introduction of the TDD philosophy spontaneous and natural.

“Forward and reverse engineering attain the same importance.” Since the model was designed upon the results of the legacy code reversing, this process, although being only a one time process, was equally important as the forward engineering.

“Constant feedback” practice was particularly important in the reengineering process since we did not have a clear idea, from the beginning, how the data reversing should be performed.

“Welcome changing requirements, even late in development.” The industrial experience report presented an iterative development, which allowed late model changes. We worked in sprints, according to the Spike principle, which implied the frequent model changes, in each sprint.

“Combine Scrum and Kanban process tools” as it is suitable. When the projects tasks can be strictly divided and planed, than the Scrum is more appropriate. But for some long running task, such as a reengineering process in our case, the Kanban was more appropriate since it was difficult to plan sprints in advance.

D. Benefits of the Agile MDD approach

We got a lot of benefits by introducing the Agile MDD approach.

1. Agile principles can make the starting curve for the MDD shorter. Through the application of the agile principles the long learning curve and introduction gap of MDD methods and tools could be avoided. Instead of spending a long time building a big thing we had a small team spending a little time building a small thing but we integrated regularly to see the whole system [11].

2. We introduced the TDD approach, showing the effectiveness of such an approach. TDD approach contributed to the accelerated reengineering to the MDD solution since the reverse and forward engineering were performed in parallel.
3. “Base your management decisions on a long-term philosophy, even at the expense of short-term financial goals.” We have prepared, in advance, for the introduction of the DMC architecture: the model specification and the reverse engineering job are already done. As well as the test cases, some of them are going to be reused.
4. The Agile MDD approach could be used instead of the full scale MDA. When all MDA recommendations could not be applied, we adjusted them to our system and organization, with a help of Agile and Lean principles.
5. Agile modeling helped against building gargantuan models [15] and specifying potentially unused data.

X. RELATED WORK

The idea of combining the agile ideas with the MDD concept has been present in both research and industry world for some time. But as a relatively new idea it is still without enough empirical results, which should lead to the right direction where and how this idea should be evolved even more. Many authors agree with the conclusion that the agile principles can be combined with the MDD, making, usually long and time-consuming process of modeling, being iterative and incremental. [34] explored this idea even more, by applying a set of agile principles on UML modeling, such as pair-modeling, test-driven development and regression testing. In [35] a comprehensive framework, showing the various ways to take advantage of the complementarity between the agile methods and MDD, is proposed.

Although it could be assumed that the agile methods should be used for the development of new software systems, they could be used for the legacy code evolution as well as discussed in [36]. By applying the agile methodology on the reverse engineering process, some authors have already made proposals for an incremental agile reverse engineering process. [37] and [38] describe a framework support for an agile reverse engineering process. [39] proposes an iterative reengineering approach that uses reverse engineering patterns for the reverse engineering and test-driven development for the forward engineering, where the reverse engineering and forward engineering activities are done independently, one after the other.

To our best knowledge there is no author who explores a simultaneous application of TDD both on the reverse engineering and forward engineering process when the legacy system is reengineered to the MDD. Additionally, there is no author who discusses the whole agile development process for the system’s evolution to the MDD solution.

XI. CONCLUSION AND FUTURE WORK

This industrial report presented an evolution process of an existing system - the architecture of the MSC definition, to a Model-Driven Development solution. The main point of this paper was to show how agile and lean principles helped us in a decision making process during the intermediate solution production, within a short time frame. In that way we coped successfully both with the management constraints as well as with the complexity and time-consuming introduction of the MDD concept.

This industrial report showed that the agile philosophy and the MDD concept can be successfully combined, resulting in an accelerated development process. Agile principles relax the OMG's recommendations reducing the complexity from the MDD concept, making the MDD easier to adopt in organizations. As this paper showed, an agile MDD could be a key success factor for organizations, which are not ready for the introduction of the full-scale MDA. Consequently we could expand Ambler's "agile modeling" philosophy on the whole MDD, including the reengineering process, meaning that it should be situational, adjusted to the running project, organizational structure and development process. Additionally, a development process based on the TDD logic can contribute to improved development efficiency and decrease the total time spent on the development and testing.

By being aware of the "Think big act small"-principle, we got a simple and applicable solution, which could easily grow to a more complex one. With a help of agile and lean ideas we modernized the MSC architecture. Such evolution made this architecture more flexible and more responsive to the future changes, regarding both the technical and the business aspects. "It is not the strongest of the species that survive, nor the most intelligent, but the ones most responsive to change [40]."

REFERENCES

- [1] Mina Boström Nakicenovic: An Agile Model-Driven Development Approach – A case study in a finance organization. Proceedings of ICSEA 2011.
- [2] M. Pikkarainen, J. Haikara, o. Salo, P. Abrahamsson, J. Still: The Impact of Agile Practices on Communication in Software Development. Journal Empirical Software Engineering, Vol. 13, Issue 3, pp 303-337, June 2008.
- [3] AgileManifesto, www.agilemanifesto.org. Accessed in May 2012.
- [4] Dave West, Tom Grant: Agile Development – Mainstream Adoption Has Changed Agility, Forrester Research, 2010.
- [5] Hans Wegener: Agility in Model Driven Software Development? Implication for Organization, Process and Architecture, 2002.
- [6] T. Dybå, T. Dingsoyr, Empirical Studies of Agile Software Development: A systematic review, Inform. Softw. Technol. (2008), doi:10.1016/j.infsof.2008.01.006
- [7] James O. Coplien, Neil B. Harrison: Organizational Patterns of Agile Software Development, Prentice Hall, 2005.
- [8] Paloma Caceras, Francisco Diaz, Esperanza Marcos: Integrating an Agile Process in a Model Driven Architecture. <http://www.sciweavers.org/publications/integrating-agile-process-model-driven-architecture> Accessed in May 2012.
- [9] Marko Boger, Toby Baier, Frank Wienberg, Winfried Lamersdorf: Extreme Modeling, 2000. <http://vsis-www.informatik.uni-hamburg.de/getDoc.php/publications/70/XM.pdf> Accessed in May 2012.
- [10] Pritha Guha, Kinjal Shah, Shiv Shankar Prasad Shukla, Shweta Singh: Incorporating Agile With MDA Case Study: Online Polling System. International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.4, October 2011.
- [11] Henrik Kniberg: Kanban Vs Scrum. <http://www.infoq.com/minibooks/kanban-scrum-minibook> Accessed in May 2012.
- [12] Oliver Sims: Enterprise MDA or How Enterprise Systems Will Be Built. MDA Journal, September 2004.
- [13] Arie van Derusen, Eelo Visser and Jos Warmer: Model Driven Software Evolution: A Research Agenda. CMSR 2007 Workshop on Model-Driven Software Evolution (MoDSE) Amsterdam 2007.
- [14] Nourchene Elleuch, Adel Khalfallah and Samir Ben Ahmed: Software Architecture in Model Driven Architecture, IEEE, pp 219-223, 2007.
- [15] Hailpern B, Tarr P: Model-driven Development: The good, the bad and the ugly. IBM Systems Journal, Vol.45, No.3, 2006.
- [16] I. Lazar, B. Parv, S. Motogna, I-G Czibula, C-L Lazar: An Agile MDA Approach For Executable UML Structured Activities
- [17] James O. Coplien, Gertrud Bjornvig: Lean Architecture for Agile Software Development, Wiley 2010.
- [18] Scott W. Ambler: Agile Model Driven Development Is Good Enough. IEEE Software 2003.
- [19] Scott W. Ambler: Agile Model Driven Development, <http://www.xootic.nl/magazine/feb-2007/ambler.pdf> Accessed in May 2012.
- [20] SunGard, www.sungard.com. Accessed in May 2012.
- [21] TNP SDK documentation: SunGard Front Arena
- [22] Mary Poppendieck, Tom Poppendieck: Lean Software Development, An Agile toolkit. Addison Wesley, 2005.
- [23] Krzysztof Czarnecki, Ulrich W. Eisenecker: Generative Programming, Addison Wesley 2000.
- [24] MDA, www.omg.org/mda. Accessed in May 2012.
- [25] Martin Fowler: Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999.
- [26] Vitaly Khusidman, William Ulrich: Architecture-Driven Modernization: Transforming the Enterprise. www.omg.org Accessed in May 2012.
- [27] Thomas O. Meservy, Kurt D. Fenstermacher: Transforming Software Development: An MDA Road Map. IEEE Software, September 2005.
- [28] S. W. Ambler, Generalizing Specialist: Improving Your IT Career Skills (2012), <http://www.agilemodeling.com/essays/generalizingSpecialists.htm>. Accessed in May 2012.
- [29] Ray Carroll, Claire Fahy, Elyes Lehtihet, Sven van der Meer, Nektarios Georgalas, David Cleary: Applying the P2P paradigm to management of large-scale distributed networks using Model Driven Approach, Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP Volume, Issue , 3-7 April 2006 Page(s):1 – 14.
- [30] Kent Beck: Test-Driven Development By Example, Addison Wesley, 2003.

- [31] Martin Fowler: Continuous integration, <http://martinfowler.com/articles/continuousIntegration.html> Accessed April 2012.
- [32] Michael Karlesky, Greg Williams, William Berezka, Matt Fletcher: Mocking the Embedded World: Test-Driven Development, Continuous Integration and Design Patterns. Embedded System Conference Silicon Valley, April 2007.
- [33] Ron Jeffries, Ann Anderson, Chet Hendrickson: ExtremeProgramming. Addison Wesley, 2001.
- [34] Yuefeng Zhang, Shailesh Patel: Agile Model Driven Development In Practice. IEEE Software 2010.
- [35] Vincent Mahe, Benoit Combemale, Juan Cadavid: Crossing Model Driven Engineering and Agility – Preliminary Thoughts on Benefits and Challenges, 2010. ECMFA 2010.
- [36] Dave Thomas: Agile Evolution – Towards The Continuous Improvement of Legacy Software. Journal of Object Technology, vol. 5, no.7, September-October 2006, pp.19-26
- [37] Maria Istela Cagnin, Jose Carlos Maldonado, Fernao Stella Germano, Paulo Cesar Masiero, Alessandra Chan, Rosangela DelossoPenteado: An Agile Reverse Engineering Process based on a Framework
- [38] Maria Istela Cagnin, Jose Carlos Maldonado, Fernao Stella Germano, Rosangela DelossoPenteado: PARFAIT: Towards a Framework-based Agile Reengineering Process
- [39] Vinicius Durelli, Rosangela Penteado, Simone de Sousa Borges, Matheus Viana: An iterative reengineering process applying Test-Driven Development and Reverse Engineering Patterns, INFOCOMP – Special Edition, p. 01–08, fev. 2010
- [40] Charles Darwin: The Origin of the Species, 1859.

Factors Leading to the Success and Sustainability of Software Process Improvement Efforts

Natalja Nikitina, Mira Kajko-Mattsson
 School of Information and Communication Technology
 KTH Royal Institute of Technology
 Stockholm, Sweden
 nikitina@kth.se, mekm2@kth.se

Abstract—Although software process improvement (SPI) may bring immediate positive results, this does not imply that the results will sustain in the long run. In order to succeed with continuous process improvement and sustain its results, organizations need to be aware of what makes their SPI efforts successful or unsuccessful. This paper presents thirty three factors that primarily contribute to the success and sustainability of SPI efforts. The factors are organized into three categories: (1) organizational factors related to the organizational structure, politics and culture, (2) implementation factors related to the planning, preparation, execution and management of the SPI projects and (3) social factors dealing with human behavior and reactions in the SPI context.

Keywords- SPI; success factors; sustainability factors; attributes; SPI status; lasting results

I. INTRODUCTION

Many software companies today invest time and resources in Software Process Improvements (SPI) in hope of increasing the effectiveness of their software processes. Despite this, not much evidence has been provided on the sustainability of the SPI results and gains [1]. Most of the reports published so far mainly give an account of the short-term gains instead. Furthermore, few authors have stated that the money and effort invested in SPI do not always lead to successful and long-lasting SPI results [2].

Even if SPI efforts show immediate gains, it is not a guarantee that the gains will be long lasting and sustainable [3]. Processes undergoing improvement may demonstrate temporary gains as a result of initial organizational enthusiasm, eagerness and/or desire to do SPI. These gains, however, may not survive in the long run. The organizations may quickly reverse to the previous software process (pre-SPI), and thereby, make the SPI efforts a waste of time and resources. The reasons for failing with SPI might be many, such as, for instance, lack of long-term management support, weakening provision of SPI resources, ill-alignment of SPI with the business goal.

To succeed with long-term SPI, organizations should continuously plan, monitor and control their SPI efforts and progress. They should continuously identify the reasons contributing to the decay or success of SPI and take appropriate measures. In this way, they will become more conscious of their SPI efforts and their opportunities and limitations which, in turn, will make them more aware of the factors contributing to the long-term success or failure of SPI.

Even though research on SPI is not new and a large number of studies have been dedicated to process improvement, there is no combined expertise on what

factors contribute to the long-term SPI success. A large number of empirical and theoretical studies list factors contributing to SPI success [2], [4-17]. None of them, however, focus on the long-term sustainability of the SPI efforts. In addition, some factors contributing to success and sustainability of SPI effort can be extracted from the case studies on SPI implementation [3], [18-30]. The studies either suggest different ways of improving processes, or they report on their experiences and lessons learnt or they discuss the incorporation of the process improvement activities into software development processes. To the authors' best knowledge, there is no study providing an exhaustive list of factors aiding software organizations in sustaining their SPI results.

In this paper, we elicit thirty three factors that primarily contribute to the success and sustainability of SPI efforts. Our goal is to create a basis for identifying factors that contribute to successful and sustainable SPI which, in turn, will aid software companies in defining, planning, monitoring and improving their SPI efforts, and in sustaining their results. This paper is an enhanced version of the previous study of the SPI sustainability success factors that have been published in [1].

The remainder of this paper is organized as follows. Section II presents background of the field. Section III describes the method used during this study. Section IV lists and provides descriptions of thirty three SPI sustainability success factors. Finally, Section V discusses the results of this study, presents final remarks and suggests future work.

II. BACKGROUND

In this section, we provide background about current software process improvement models. Section II.A describes general and common stages of software process improvement process. Section II.B presents process capability and maturity assessment models to be used in SPI. Section II.C presents software process improvement approaches existing today. Finally, Section II.D lists software development methods that incorporate the practices of process improvement.

A. Software Process Improvement

Software process improvement is a set of SPI activities leading to an improved software process quality, and thereby, to an improved software product quality [31]. Each SPI effort is unique in its design. It strongly differs with respect to the individual and cultural characteristics and needs of an organization and the status of the processes undergoing improvement [32], [33]. For this reason, it may be difficult to suggest a process model that is suitable for all kinds of SPI contexts.

Many researchers have proposed high level SPI models, such as [31], [34-36]. These models differ in their designs. Nevertheless, they have defined common cyclic stages that are believed to bring maximum benefit to the improving process and business. The most accepted representation of the continuous SPI is illustrated in Figure 1. Its cyclic phases are:

- *Plan (SPI planning)*: aiming at defining process improvement goals and vision, as well as identifying process improvement activities and creating a process change plan.
- *Do (Process change)*: aiming at changing the process according to the planned improvement.
- *Check (Process review)*: aiming at assessing/reassessing/measuring the process according to the goals of the process improvement, analyzing the process and its measurements and comparing them to the expected results.
- *Act/Adjust (Process adjustment)*: aiming at requesting the corrective actions in order to reach planned results and determining the weaknesses and potential improvements of the SPI process.

B. Process Capability and Maturity Assessment Models

There is a large amount of process maturity models that have been designed to help software organizations to assess the status of their software processes and identify the areas of future improvement. The best known ones are CMM (Capability Maturity Model) [37], CMMI (Capability Maturity Model Integration) [38] and SPICE (Software Process Improvement and Capability dEtermination) [39].

Both CMM and its improved version CMM Integration (CMMI) is a standardized framework for process assessment and improvement [37] [38]. Both models provide software organizations with a roadmap for process standardization and improvement. Their frameworks are based on the implementation of the key practices within certain key process areas according to the set improvements goals and the desired maturity level. CMM and CMMI define five maturity levels for process assessment. The maturity levels of CMMI are: *Initial, Managed, Defined, Qualitatively Managed* and *Optimizing* [38]. CMM has a *staged* representation implying that the improvements are targeted to increase the company's overall capability/maturity level. CMMI, on the other hand, in addition to the *staged* representation also includes

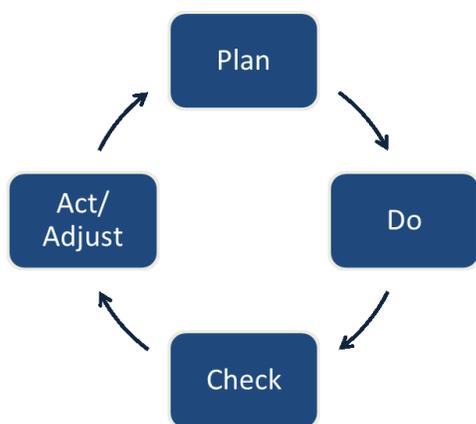


Figure 1. Continuous Software Process Improvement Cycle

the *continuous* representation implying that improvements are focused on companies' specific target process areas. With this structure, CMMI aims to help software organizations to assess their organizational maturity or process area capability, establish priorities for improvement, and implement these improvements [38].

SPICE, also known as ISO 15504, is an international reference model for process assessment and improvement. The model can be used for process assessment and its capability determination. In the similar way as CMMI, SPICE is organized into six capability levels that characterize the process as: *Incomplete, Performed, Managed, Established, Predictable* and *Optimizing*. The capability levels, in turn, consist of process attributes, which further consist of generic practices. SPICE model provides tools for standardized process assessment and suggestions for defining process maturity. [39]

C. Software Process Improvement Approaches

Other process improvement approaches or methods can be used as an addition to or as an alternative for capability maturity models. The most commonly used SPI approaches are SixSigma [40] and IDEAL [41].

SixSigma is business management approach for improving engineering and development processes. It is a disciplined data driven approach aiming at improving a development process by identifying and removing the causes of product defects. Using a measurement-based strategy, SixSigma defines how the process is performing and how it should be improved. The improvement of the existing process is done by following five iterative steps: 1) define the defects and project goals, 2) measure the process attributes with respect to its quality and efficiency, 3) analyze the process and determine the root causes of defects, 4) improve the process by eliminating the defined defects, and 5) implement control mechanisms for sustaining the achieved improvements. [40] [42]

IDEAL is a process improvement implementation model that has been primarily designed for supporting the implementation of CMM and CMMI maturity models [43]. It encompasses five stages of a process improvement cycle. Those are the following: 1) *Initialize*: start the improvement program, 2) *Diagnose*: assess the current state of the process, 3) *Establish*: set the implementation strategy and improvement program, 4) *Act*: implement process improvements, 5) *Leverage*: analyze the improvement effort and revise the approach. [34] [41]

D. Software Development Methods that Contain SPI Practices

Many of the software development methods incorporate process improvement activities into their development activities. The best known ones are Lean and Scrum.

Lean software development guides organizations on how to deliver increments of real business value in short time boxes, by means of optimizing/improving their software processes [44]. In the core of Lean software development, there are seven principles aiming at continuous improvement of the process based on the identification and elimination of the inefficiencies (waste) in the process [45]. The core principles are the following: 1) eliminate waste, 2) amplify learning, 3) decide as late as

possible, 4) deliver as fast as possible, 5) empower team, 6) build in product quality, and 7) see the whole [45].

Scrum is a well-known iterative, incremental, light-weight and agile method for software project management. It is most often used in small or medium-sized development organizations. The method is focused on managing software development projects by means of strictly defined: 1) roles, such as scrum master, product owner and the team, 2) meetings, such as daily standups, release and sprint planning meetings, retrospectives and demos, and 3) process artifacts, such as product and sprint backlog [46]. Scrum incorporates in itself continuous process reviews that are done at the end of each development iteration, called *sprint*. This contributes to the light weighted continuous process improvement [32].

III. METHOD

In this section, we present our research method. We first present the research steps in Section III.A. We then describe the questionnaire used in one of the research steps in Section III.B. Finally, in Section III.C, we describe the validity of our results.

A. Research Steps

The overall research consisted of the three following steps: (1) *Literature Study*, (2) *Empirical Study*, and (3) *Data Analysis*.

During the first two steps, we elicited SPI sustainability success factors, first by reviewing literature and then by interviewing software practitioners. These two steps were conducted independently. This implies that the results of the first step did not constitute input to the second step, and vice versa. In the third step, we combined and analyzed the results as achieved in the first two independently done steps. Below, we briefly describe the three steps.

1) Literature Study

During the literature study, we reviewed more than 45 publications dealing with SPI projects. These were mainly experience reports and case studies that had been retrieved from IEEE, ACM, Springer, John Wiley and Sons, and other publishers. Out of them, we chose 27 empirical reports describing conditions contributing to or subtracting from the success of SPI projects [2-27], [30]. Our goal was to draw out factors that contributed to the sustainability of SPI efforts.

The majority of the publications studied mainly reported on the empirical process improvement projects. They did not focus on outlining the conditions contributing to the success of SPI efforts. However, some of the conditions could be indirectly recognized out of their contexts and results. Only three publications provided direct and explicit feedback on critical SPI success factors. These were [4-6].

Based on the literature studied, we drew out factors that were critical for a successful initiation and implementation of SPI, and successful preservation of its results. This step resulted in a preliminary list of SPI sustainability success factors. Having this list as a basis, we reviewed the publications anew, now with the purpose of studying their explicit and implicit descriptions of the success factors, their contexts, and impact on the sustainability of the SPI efforts. This step resulted in twenty seven SPI success factors.

TABLE I. INTERVIEW QUESTIONNAIRE

- | |
|---|
| <ol style="list-style-type: none"> 1. Are you aware that the information you provide will be kept confidential? 2. Have you been involved in process improvement or process transition before? To what extent? <ol style="list-style-type: none"> a. If yes, have the results of the process improvement been lasting? <ol style="list-style-type: none"> i. If yes, why do you think the results have been lasting? ii. If no, why do you think the results have not been lasting? 3. What factors contribute to the process improvement sustainability? Please list them and motivate your answers. 4. What factors prevent the process improvement sustainability? Please list them and motivate your answers. 5. What are your suggestions for keeping the process improvement results lasting/sustainable? Please list them and motivate your answers. |
|---|

2) Empirical Study

During the empirical study, we interviewed 45 software engineers who had been involved in or who had been affected by SPI projects. Among the interviewees, there were twenty seven software developers, ten testers, seven development managers and one SPI manager. They came from twelve different medium-sized software organizations, located in Sweden (23 participants), Vietnam (18 participants), Bangladesh (2 participants), China (1 participant) and Island (1 participant).

Each interviewee was interviewed only once, in a tête à tête manner. Twenty seven interviews were recorded and then transcribed. The other eighteen interviews were not recorded due to the fact that the interviewees felt ill at ease to be recorded. However, the interviews with them were thoroughly documented during and after each interview. On average, the individual interviews lasted for forty minutes.

All the interviews were analyzed using the hermeneutics approach [47]. The SPI success factors listed by the interviewees were analyzed and grouped together when concerning the same or similar issues. The factors that were mentioned by more than one interviewee were joined together and given a common name. The factors that were mentioned by only one interviewee were not included in this paper at all.

3) Data Analysis

During the *Data Analysis* step, we analyzed the results of the literature study and the empirical data using the hermeneutics approach [47]. Here, we identified and analyzed the sustainability factors as drawn out in the former study steps (the literature and empirical study steps). In total, we identified thirty three SPI sustainability success factors, out of which twenty four factors were commonly identified in both literature and empirical studies. Out of twenty seven success factors identified in the literature, three were not confirmed during the empirical study. In addition, six out of the thirty factors identified during the empirical study had not been identified in the literature studied.

Finally, we combined all the elicited factors, organized them into categories and put them into a list of SPI sustainability success factors. It is this list that constitutes

the body of this paper and a foundation for the creation of the SPI health attributes in [48].

B. Questionnaire

When educating knowledge about the SPI success factors, we used open-ended and semi-structured interviews [47]. This helped us encourage our interviewees to provide additional information that might be found useful for understanding the factors.

The interview structure was based on the questionnaire presented in Table I. Its questions were aimed at identifying both success and failure factors. Therefore, the questionnaire was structured into the following five groups of questions: (1) reasons for why SPI efforts have been lasting, (2) reasons for why SPI efforts have not been lasting, (3) factors contributing to the SPI sustainability, (4) factors preventing the SPI sustainability, and finally, (5) suggestions for how to keep the SPI efforts sustainable.

C. Validity

All the qualitative research methods encounter validity threats [47]. Those threats concern *construct validity*, *internal validity* and *external validity*.

Construct validity refers to the degree to which inference can be made from the operational definition of a variable to the theoretical constructs [49]. The main threat to *construct validity* is to guarantee that the right measures have been chosen for the study. Here, the risk was that we might use wrong measures, and as a result, we might misinterpret the SPI sustainability success factors. To minimize this threat, we conducted both theoretical and empirical studies. Moreover, we employed the multiple sources of data during the empirical study by interviewing different roles in twelve different organizations.

Internal validity refers to the degree of inferences of the cause-effect or causal relationships in the study [49]. The main threat to *internal validity* for the literature study was the fact that we might misinterpret the conclusions presented in the literature or use too few literature sources. Therefore, in this study, we first made a comprehensive search in various scientific sources out of which we extracted 27 experience reports. The main *internal validity* threat for the empirical study was that the interviewees might have misunderstood the impacts on the SPI sustainability. To minimize this threat, we used various roles involved in SPI in different software organizations.

External validity refers to the degree of whether the sample findings can be generalized [49]. The main *external validity* threat to our empirical study was the fact that the SPI sustainability factors that had been identified during the interviews were based on the experiences of only 45 individuals who belonged to medium-sized software companies. Therefore, we believe that the empirical findings of this study should be found more useful for medium-sized software companies. Nevertheless, by incorporating them with the results of the literature study, we are confident that our findings and conclusions are useful for all sizes of software companies whether large, medium-sized or small.

IV. SPI SUSTAINABILITY SUCCESS FACTORS

In this section, we present the identified factors that lead to the success and sustainability of the SPI efforts.

Some of them have direct influence over the SPI success and sustainability whereas some other factors have an indirect influence. Therefore, when describing them, we state their influence wherever it is relevant.

Based on both literature and empirical studies, we have identified thirty three SPI sustainability success factors. During the literature study, we have identified twenty seven SPI success factors, out of which twenty four factors overlapped with the factors that have been identified during the empirical study. The interviews have additionally resulted in six new SPI factors.

Just because these two studies were done independently, they had led to two groups of SPI success factors: (1) the ones that are common to the two studies, and (2) the ones that have been elicited within one type of a study but not within the other. When describing them in this section, we clearly identify their sources. Additionally, we list them and their sources in Tables II, III and IV.

To facilitate our presentation, we group the elicited SPI sustainability success factors into three categories as defined in [18]. These are *organizational factors*, *implementation factors* and *social factors*. The factors are described in Sections IV.B, IV.C and IV.D that follow Section IV.A, which briefly presents the core SPI roles that may vary in different academic and industrial contexts.

A. Roles

Many different roles are involved in process improvement. Their naming and responsibilities vary in different academic and industrial contexts. For this reason, we identify and define the following roles involved in SPI:

- *Stakeholder*: a person or a group that is involved in or affected by SPI. Stakeholders include all the internal and external roles that are influenced by SPI.
- *Technical staff*: a group consisting of developers, testers, development managers, support personnel and other roles involved in executing the process undergoing the improvement. They are the “doers” within the process undergoing improvement, and therefore, they get affected by the process change the most.
- *External SPI leader*: a person or a group that is in charge of the overall SPI process. He/she initiates the improvement projects, requests resources, encourages local improvement efforts and establishes communication channels between different groups. External SPI leader is not the doer in the process to be improved. For this reason, he/she is seen as an external and independent role.
- *Internal SPI leader*: a person or a group within the technical staff who is responsible for supporting and following the SPI strategy on a local process level.

B. Organizational Factors

Organizational factors are critical success factors that are related to the organizational structure, politics and culture [18]. They have a substantial impact on the SPI effort and its sustainability.

Table II shows nine organizational factors. We have grouped them into three clusters: 1) *Support of SPI* focusing on the management support and sponsorship of the SPI project, 2) *Resources* targeting people resources required for conducting the SPI project, and 3) *Alignment*

aiming at aligning SPI with the organization-related factors. The organizational SPI success factors are following:

1) *Management continuously supports and commits to the SPI process*

To provide long-term sustainable results, software improvement requires continuous investment in time, resources and effort. This, in turn, requires that management is strongly committed to and continuously supports the SPI efforts [6], [7], [18], [20], [28]. Strong management commitment helps retain high priority of the SPI projects and the continuous management support helps assure continuous supply of the required resources. It is especially important in the initial SPI phases during which the cost of the SPI activities is higher than the initially expected and planned cost [4].

Even our interviewees have stated that SPI projects need investment in time and resources in order to achieve sustainable results. According to them, this cannot be achieved without commitment and support of top management. Without management support, the SPI effort and results are doomed to decay. Our interviewees have also pointed out that management should prioritize the SPI activities and assign resources to them. This will prevent the SPI activities from getting neglected.

2) *Resources are dedicated to SPI*

Resources that are fully or partly dedicated to the SPI activities are the most important organizational SPI success factor. As many as 72% of SPI improvement projects have suffered from lack of resources and constant time pressure [4], [8], [19], [26].

According to the literature studied, SPI projects need to have dedicated time and resources. SPI projects cannot run on their own. Investment in resources has been recognized not only for starting and implementing the SPI projects but also for sustaining the achieved results [4], [5], [8], [9], [11], [18], [21], [26].

Our interviewees were of the same opinion. According to them, without dedicated resources, SPI can only rely on the engagement of individuals. The engagement however tends to decrease with time. Therefore, to guarantee the sustainability of the SPI efforts, it is important to dedicate resources to both the SPI and to the process undergoing the SPI.

3) *SPI responsibilities are clearly specified and compensated*

Clarity in the definition of the SPI roles and their responsibility assignments are very important. According to the literature studied, people involved in SPI should have clear responsibilities and compensation for their effort [4], [5]. If they are assigned to the SPI related tasks, they should be relieved from other tasks. Time dedicated to the SPI activities should be compensated in the same manner as other work. Otherwise, the SPI activities may be done in a rush, they may be neglected, they may be delayed or they may even be forgotten.

Our interviewees were of the same opinion. According to them, process related problems often start when no one is responsible for the process.

4) *Competent external SPI leaders are designated*

According to the literature studied, the level of competency, experience, commitment and engagement of the external SPI leaders can greatly determine the success of the SPI projects [4-6], [18]. However, as [4], [5] claim, this may not always be enough. Authority and respect paid to the external SPI leaders is just as important. Even if the SPI leaders are in a privileged position, it still does not imply that they have high enough authority, trust and respect among the technical staff members. If so, then their ideas may not be supported and successfully transmitted to the process change [4-6], [18]. Trust and respect may only be gained via personal qualities such as honesty, credibility, reliability, experience, reputation and good leadership.

TABLE II. ORGANIZATIONAL FACTORS

Cluster	SPI sustainability success factor	Recognized in literature	Recognized by No. of interviewers
Support of SPI	1. Management continuously supports and commits to the SPI process	[4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [16], [17], [18], [20], [21], [22], [23], [24], [25], [26], [27]	18
	2. Resources are dedicated to SPI	[3], [4], [5], [8], [9], [11], [16], [18], [19], [20], [21], [22], [23], [25], [26], [27]	18
Resources	3. SPI responsibilities are clearly specified and compensated	[4], [5], [10], [11], [16], [17], [19], [27]	14
	4. Competent external SPI leaders are designated	[4], [5], [6], [9], [10], [11], [12], [13], [16], [18], [20], [23], [27], [30]	22
	5. Internal SPI leaders are designated	[3], [6], [7], [10], [12], [13], [18], [20], [21], [23], [24], [27]	6
	6. The level of technical staff turnover is low	-	4
Alignment	7. SPI is aligned with business goals	[12], [13], [16], [18], [21], [22]	-
	8. SPI is aligned with organizational policies and strategies	[4], [16], [17], [19], [21], [26]	-
	9. SPI methods are tailored to specific organizational contexts and needs	[3], [6], [17], [18], [22]	11

The importance of the external SPI leaders was also raised during the interviews. According to our interviewees, to make the SPI results last, there should be an external SPI leader, a person or a group of people who have knowledge of SPI and who take on the responsibility of driving it. The external SPI leader should lead the people and guide the process towards a continuous and sustainable improvement.

5) *Internal SPI leaders are designated*

Since external SPI leaders and managers are not directly involved in the development process, it is important to have internal leaders as well. According to the literature studied, the internal SPI leaders are recognized as *important SPI actors* since they take on the immediate responsibility for leading and supporting continuous process improvement [6], [7], [20], [21]. By possessing knowledge of the process, they are able to adapt the improvement suggestions to the different needs of the development teams, projects and cultures. They help SPI activities get started and their engagement aids in winning support of their team members towards SPI [20].

The importance of designating internal SPI leaders was also recognized during the interviews. According to our interviewees, the involvement of the internal SPI leaders helps spread commitment to the process and create strong process ownership. Internal leadership creates continuous control that the development process is followed in a correct way and that the technical staff is engaged in SPI.

6) *The level of technical staff turnover is low*

According to our interviewees, high people turnover can become a significant barrier to the sustainability of the SPI efforts. When the key employees leave the company, so does the knowledge of the process and SPI. With a high technical staff turnover, more effort needs to be spent on the education and training of the new hires.

7) *SPI is aligned with business goals*

The goals of SPI projects should not only go in line with the standardization of the process and quality standards, but also with the business goals of the company. According to the literature studied, alignment of SPI goals with the organizational business goals contributes to the better management of, commitment to and support of the SPI projects [12], [13], [18], [21], [22].

8) *SPI is aligned with organizational policies and strategies*

Improvement projects often conflict with the existing organizational policies by requiring changes to the routines and processes that are common to the whole organization. Therefore, as stated in the literature studied, organizational policies have to be aligned with the SPI goals and vice versa.

In cases when organizations do not have any policies, they have to establish ones and make the process standardization and improvement coherent with them. Lack of organizational policies to support process changes can potentially become a big barrier for a successful process improvement [4], [19], [21], [26].

9) *SPI methods are tailored to specific organizational contexts and needs*

Each organization is different with respect to its structure, culture and policies. For this reason, as stated in the literature studied, SPI initiatives should consider the contextual specifics of the organizational culture, product

characteristics, customer availability and people influenced by the process. The adaptation of process improvement methods to the specific organizational contexts and needs helps address individual problems and contributes to sustainable SPI efforts [6], [18].

The interviews have led to the same conclusion. According to our interviewees, if the SPI is not aligned with the organizational needs, or if it does not fit the established organizational and national culture, then it is more difficult to win people's support and commitment. Moreover, the people will resist process changes and the results achieved by SPI will be easily lost.

C. *Implementation Factors*

We have elicited fourteen implementation factors. They are all related to the planning, preparation, execution and management of the SPI projects. As shown in Table III, they are grouped into four clusters: 1) *Education and knowledge* focusing on training and expertise of stakeholders, 2) *SPI strategy* targeting preparation and vision of the SPI project, 3) *SPI management and execution* dedicated to issues related to management and execution of the SPI project, and 4) *Continuity of SPI effort* focusing on the mechanisms for enabling the continuity of the SPI effort. The factors are the following:

1) *Stakeholders are trained in software process*

Process improvement often implies changes to the process in form of introduction, removal or modification of new techniques and practices. Hence, as pointed out in the literature studied, the technical staff needs to be trained in the process and its techniques and practices in order to fully understand their role in the process change. They need to be prepared for the process improvement and understand the reasons behind each suggested change. Moreover, other stakeholders that are affected by SPI should also receive necessary training. Otherwise, they will less likely follow the new process [19]. For this reason, it is needed to train the stakeholders in the new process, new techniques and practices not only for supporting the implementation of process changes but also for sustaining improvement results. In organizations or cultures where knowledge of the process is low, the training in the process is even more important [26]. The levels of training may differ from stakeholder to stakeholder with respect to the stakeholders' training needs and their level of involvement in SPI.

The need for adequate process training was also raised during the interviews. According to our interviewees, all the company employees need to have necessary training in the new method in order to understand it and to be able to follow it properly and dedicatedly. The process training increases employee motivation in the SPI and decreases resistance to process change.

2) *Stakeholders are continuously mentored and coached*

Training in the new process contributes to its understanding and allows stakeholders to follow it dedicatedly. Still however, according to the literature studied, training in the software process may not be enough. Some stakeholders may misunderstand the process or continue following old techniques and practices. Therefore, the stakeholders should be mentored and coached in SPI and the process changes. [7], [23].

TABLE III. IMPLEMENTATION FACTORS

Cluster	SPI sustainability success factor	Recognized in literature	Recognized by No. of interviewers
Education and knowledge	1. Stakeholders are trained in software process	[3], [4], [7], [8], [9], [11], [18], [19], [20], [21], [22], [23], [24], [25], [30]	31
	2. Stakeholders are continuously mentored and coached	[3], [7], [8], [9], [23], [25]	19
	3. SPI leaders possess experience and expertise in SPI	[7], [8], [9], [16], [18], [19], [21], [22], [25], [26], [27], [30]	17
SPI strategy	4. SPI goals and objectives are clear and realistic	[4], [5], [6], [12], [13], [16], [17], [18], [19], [21], [22]	15
	5. SPI method is well defined	[7], [8], [9], [16], [17], [18], [20], [24], [25], [26]	-
SPI management and execution	6. SPI project is effectively managed	[6], [8], [9], [16], [19], [25], [26], [30]	16
	7. Process improvements are focused on specific areas	[20], [21], [22], [27]	2
	8. Process improvement effort is flexible	-	11
	9. Information about SPI activities and its results is disseminated	[6], [9], [10], [12], [13], [17], [19], [22], [26]	26
	10. Process standards are defined and enforced	-	17
	11. SPI effort brings positive results	-	18
Continuity of SPI effort	12. Software process is monitored and measured	[3], [6], [7], [12], [13], [17], [20], [21], [23], [27]	21
	13. Software process and its efficiency are continuously reviewed	[2], [3], [7], [8], [10], [12], [13], [22], [23], [24], [25], [27]	13
	14. SPI effort is continuous	[2], [8], [11], [14], [27]	23

According to our interviewees, the internal SPI leaders and other stakeholders responsible for the improvement activities have to be coached by the experienced external SPI leaders on how to implement improvements and how to follow the new process. Continuous mentoring and coaching increases the credibility of the strategic SPI decisions and contributes to building trust both in those decisions and in the new process.

3) SPI leaders possess experience and expertise in SPI

Process improvement implies changes to the deeply ingrained organizational culture, habits, working patterns and manners that have been developed throughout a long time. To change them is very difficult. According to the literature studied, however, it is easier to change them if the SPI team possesses enough knowledge and experience in implementing software process improvement changes. If there is lack of such knowledge and experience, then there is a risk of using unsuitable SPI strategy and of having poor SPI execution, which could potentially fail the SPI projects [7-9], [19], [25], [29].

Our interviews have also led to the same conclusion. The interviewees have mentioned the importance of the experience and expertise to be possessed by the SPI leaders.

4) SPI goals and objectives are clear and realistic

SPI projects should have clearly specified goals and objectives. Our literature study shows that clear, realistic and well communicated SPI goals contribute to good understanding of the SPI process and assurance that they are well understood across all the organizational levels [5].

Realistic SPI goals lead to realistic expectations and aid in maintaining high motivation for and support of the SPI activities. Unrealistic, too ambitious or unreachable objectives, on the other hand, may jeopardize the SPI projects, by decreasing employees' engagement and motivation even in projects with positive results [4], [21]. Our interviews have led to a similar conclusion.

5) SPI method is well defined

Software process improvement is a complex and time consuming process. Following a well defined and structured SPI implementation method strongly contributes to its success [7], [9], [25]. According to the literature studied, the SPI method should be suitable to the organization, its size and goals.

6) SPI project is effectively managed

Management of the SPI project involves a wide range of activities such as planning for change, identifying actors involved, ensuring the level of understanding the process changes, monitoring the status of SPI, evaluating the progress, and the like. It needs to be performed in an effective and professional manner [26]. According to the literature studied, without project management, the SPI project is doomed to fail or it may lead to chaos [6].

Our interviews have pointed out that effective management and execution of SPI are key elements of the successful SPI effort and its sustainable results.

7) Process improvements are focused on specific areas

At the beginning of the SPI projects, companies can be overwhelmed with the amount of suggestions for the improvements. Such being a case, as stated in the literature

studied, it is important not to do too many changes at once. Instead, companies should focus on a few specific process areas and a few process improvement goals. They should also prioritize the SPI suggestions and implement only one or few improvements at a time [20], [21]. This leads to easier and more efficient implementation, control, measurement, and thereby, to more sustainable results of SPI efforts.

Our interviews have led to the same conclusion. The interviewees added that process changes should be introduced in a slow manner and supported by training sessions. This can contribute to the understanding of the process undergoing change and of the impact of the process changes.

8) *Process improvement effort is flexible*

Software process should continuously change and adapt to the organizational needs and situation. SPI activities can be risky and may not always lead to the expected results. Therefore, according to our interviewees, process improvements should be flexible and allow for experimenting with the process. In cases when the process change is proven to be unsuccessful or unsuitable, the organization should be able to quickly rollback the process to the pre-change status.

9) *Information about SPI activities and its results is disseminated*

SPI projects bring many changes to the process and daily routines. These changes have to be communicated to all the stakeholders that can be directly or indirectly impacted by the changes. According to the literature studied, insufficient communication of the SPI changes may lead to lack of transparency of the SPI projects, confused personnel and poor quality process. Team collaboration and communication, on the other hand, may help the staff members to exchange knowledge and experience during the improvement projects and contribute to a more coherent organizational culture [6].

The need for communicating on the SPI activities and their results has also been raised by our interviewees. According to them, sufficient communication positively impacts motivation in SPI and acceptance of the new process changes.

10) *Process standards are defined and enforced*

In some companies, the newly introduced process can just run by itself. Its main fuel is primarily high commitment and engagement of the technical staff. However, in companies that have low commitment towards or poor understanding of the development process, people are tempted to disregard the process standards, unless there is a strong control mechanism in place [34]. Even when properly trained, the staff may not follow the newly introduced process. Therefore, as stated in the literature studied, in order to guarantee that the process is dedicatedly followed by all the stakeholders, it should be enforced and controlled by the SPI managers [34].

Our interviews have led to a similar conclusion. The interviewees have also suggested that the employees that are not following the software process procedures correctly should be informed and consequently corrected. The interviewees also highlighted the importance of accessible and updated software process documentation.

11) *SPI effort brings positive results*

As mentioned before, the results of the SPI activities should be disseminated to all the stakeholders. However, as discovered during the interviews, just the dissemination of the SPI results is not enough. The results achieved by the early SPI effort should be positive and should speak for themselves.

Early gains of SPI effort can encourage and motivate stakeholders to continue with the SPI activities and can change the opinions of those who did not support SPI from the very beginning.

12) *Software process is monitored and measured*

Continuous process monitoring and measurement indicates whether the SPI activities are effective or not, and allows to provide early feedback on the sustainability of the SPI efforts. Hence, as stated in the literature studied, it is important to evaluate and measure the process on a continuous basis in order to reinsure its purpose and to increase the engagement of the SPI supporters. Measured and acknowledged process improvement will positively affect team morale and motivation [7], [12], [13], [20].

Our interviewees have also stated that measurement and evaluation of the SPI results can positively impact the engagement in and motivation for future SPI.

13) *Software process and its efficiency are continuously reviewed*

To achieve continuous process improvement, the SPI process and its efficiency should be reflected on and evaluated on a continuous basis. As stated in the literature studied, process reviews, such as retrospectives, allow learning from previous experience and from experimenting with the process, which, in turn, contributes to a self-driven continuous process improvement, and thereby, to long lasting SPI results [12], [13].

Our interviews have led to the same conclusion. According to our interviewees, process reviews help to identify problems in the current process and to acknowledge benefits achieved by SPI. This, in turn, significantly contributes to the sustainability of the achieved results. Without frequent reviews and changes to the process, gains of SPI will soon outdate.

14) *SPI effort is continuous*

Software organizations have dynamic and continuously changing structures. Organizational culture, availability of the customer and background of the employees are always changing. Hence, a static process that is not improving or adapting to the changing organizational needs is failed to decay [34]. Moreover, the results of the SPI efforts will be lost if the organization will stop improving its process.

To sustain the gains of the process improvement efforts, the company should view the SPI as a continuous activity. According to the literature studied, continuous SPI effort cannot be achieved without mechanisms for continuous process review and tuning [34], and comprehensive support of those responsible for the process [6]. In addition, all the roles responsible for the SPI project should continuously reaffirm commitment to change, communicate progress of improvement, and provide continuous feedback and motivation [6].

Our interviews have also confirmed that time and money should be continuously invested into the SPI effort in order to maintain its results.

D. Social Factors

We have identified ten different social SPI sustainability success factors. Social factors deal with human behavior and reactions in the SPI context. As shown in Table IV, they are grouped into three clusters: 1) *Understanding and awareness of SPI* focusing on common understanding and awareness of the SPI, 2) *Attitude to SPI* targeting stakeholders' attitude towards SPI, and 3) *Facilitation of SPI* listing factors that may increase stakeholders' motivation in SPI. The social SPI success factors are the following:

1) *Stakeholders have a common understanding of the process undergoing change*

The process cannot be efficiently improved unless it is properly understood. According to the literature studied, the technical staff and management have to reach consensus on the status of the current process, its problems and possible solutions, as well as the organization's vision and improvement goals [6]. Common understanding of the current and new process, suggested changes and its potential benefits are important to increase support for process improvement among all the stakeholders involved.

Our empirical study has led to the same conclusion. According to our interviewees, all the stakeholders should understand the reasons behind process changes. An important stakeholder here is the technical staff who has to change the previous habits and adapt to a new way of working. Our interviewees have also pointed out that common understanding of the new process, of the SPI activities and their potential benefits strongly contribute to the increase of commitment and motivation towards SPI.

2) *Stakeholders are aware of complexity, challenges and benefits of SPI*

Since SPI requires continuous effort and often brings mainly long-term results, it is important that everybody involved in it is aware of its complexity, challenges and

future benefits. Hence, according to the literature studied, organizations must make sure that all the stakeholders involved are aware of complexity and potential benefits of SPI. This can be realized via education, training and effective communication. Raising awareness of SPI and effective communication of its complexity, challenges and benefits strongly affects the success of the SPI projects [8-10], [23-26].

Our interviews have also shown that the stakeholders need to understand the reasons behind SPI and its potential benefits in order to accept and commit to the process change.

3) *Stakeholders have realistic expectations*

Our interviews have indicated that in order to be satisfied with SPI and its results, the employees affected by SPI should have realistic expectations. Otherwise, the stakeholders will get disappointed with SPI and will not continue with it, even though SPI brings positive results.

4) *Technical staff accepts SPI activities*

Changes to the process may affect daily work of many employees. Therefore, according to the literature studied, it is important that all the members of the technical staff agree and accept future process changes [18], [27]. This can substantially decrease inertia to change. Acceptance of process changes can be encouraged by high involvement of the technical staff in the SPI activities.

Our interviews have also led to the same success factor. According to our interviewees, if all the personnel accept the newly changed process, then there is a greater opportunity that the changed process will be sustained. Mutual acceptance of the changed process and SPI activities is a key to sustain the results achieved by the SPI.

5) *Technical staff is committed to the SPI process*

Acceptance of SPI activities is a critical success factor when starting SPI projects. According to the literature studied, however, it needs to be complemented with the

TABLE IV. SOCIAL FACTORS

Cluster	SPI sustainability success factor	Recognized in literature	Recognized by No. of interviewers
Understanding and awareness of SPI	1. Stakeholders have a common understanding of the process undergoing change	[6], [11]	17
	2. Stakeholders are aware of complexity, challenges and benefits of SPI	[8], [9], [10], [17], [20], [23], [24], [25], [26]	4
	3. Stakeholders have realistic expectations	-	4
Attitude to SPI	4. Technical staff accepts SPI activities	[3], [16], [18], [27]	24
	5. Technical staff is committed to the SPI process	[3], [16], [21], [22], [23], [24]	10
Facilitation of SPI	6. Stakeholders are being encouraged to support SPI	-	16
	7. Technical staff is rewarded for contribution to SPI success	[7], [9]	4
	8. SPI leaders encourage initiative and openness of stakeholders	[11], [12], [13], [19]	2
	9. Technical staff participates in SPI	[3], [4], [5], [6], [8], [9], [10], [12], [13], [18], [21], [23], [24], [25], [27], [30]	22
	10. Technical staff owns the software process	[3], [5], [7], [18], [27]	20

commitment of the technical staff. Commitment to the SPI projects is inevitably another significant success factor to sustain the results of the SPI projects.

Management commitment to SPI projects has already been listed as a significant SPI success factor. However, commitment of technical staff is just as important [21-24], [28]. Together with the increased motivation and engagement, the commitment of the technical staff can become a driving wheel of process improvement [3]. Committed staff takes on the responsibility and ownership of the process and keeps process in a healthy state [3].

Commitment of the technical staff has also been educated during our interviews. Our interviewees have stated that if the company personnel does not commit to the process changes, it will most likely go back to the pre-SPI process state.

6) *Stakeholders are being encouraged to support SPI*

Commitment to and support of SPI by all the stakeholders is a great asset to help successful SPI implementation and to decrease inertia towards change. However, it is not easy to reach everybody's support of SPI. Therefore, our interviewees suggested that during the early stages of the SPI project, the management should start encouraging the stakeholders towards supporting SPI. Encouraging the technical staff in SPI from the very beginning would increase support, motivation and engagement in future SPI activities.

7) *Technical staff is rewarded for contribution to SPI success*

Moral appreciation and financial rewarding acknowledge individual contributions to SPI. Recognized contribution engages and motivates people to continue with the SPI effort [7], [34]. The technical staff should also be rewarded for showing interest in and for contributing to the process improvement activities.

According to our interviews, the organization should celebrate each SPI success and reward personnel for their contributions. This will increase overall motivation and commitment to SPI.

8) *SPI leaders encourage initiative and openness of stakeholders*

To be able to suggest future process improvements, the weaknesses and problems of the current process need to be continuously identified. Some of those weaknesses and problems can relate to specific individuals and fulfillment of their responsibilities. Therefore, it is important to focus on identifying the process weaknesses and not on playing *blame games* [19], which can only lead to frustration and inertia towards process change [11], [19].

According to the literature studied, SPI leaders should focus on process weaknesses and problems and should encourage initiative, innovation, creativity and openness in stakeholders involved. Without it, employees cannot share valuable ideas, and thereby, contribute to process improvement [11]. Few of our interviewees were of the same opinion.

9) *Technical staff participates in SPI*

Technical staff constitutes an important process knowledge and experience asset [6]. By knowing all the nooks and crannies of the process, the staff may provide useful feedback on the suggested SPI changes [27]. For this reason, it is important that they are involved in

identifying process pains and in suggesting solutions for them [6], [8], [9], [12], [13], [18], [25].

The literature findings show that the involvement and participation of the technical staff reduce resistance to change, and thereby, strongly impact the SPI success [6], [18], [21]. By being involved in the SPI activities, the technical staff members feel more motivated to adhere to the process changes, and therefore, they are more likely to accept them [6], [18]. If, on the other hand, they are not convinced, then the process improvement projects will have small chances to succeed. Technical staff involvement was found especially important in immature organizations [25].

Many of our interviewees have also mentioned that the involvement of the technical staff contributes to the alignment of SPI methods to the organizational needs. It also decreases inertia towards change and increases motivation, and thereby, it significantly affects the sustainability of the SPI efforts.

10) *Technical staff owns the software process*

Disregarding the reasons behind the SPI projects, the new process has to be accepted and followed by the team. According to the literature studied, it is important that not only external and internal SPI leaders but also all the technical staff members take on the ownership of the process to be improved. The members should take the responsibility for tailoring the process and for continuously improving it. It is only in this way they will feel more affiliated with the process and more responsible for future process improvement. This, in turn, will lead to a built-in, self-driven continuous process improvement process, which, in turn, will strongly contribute to the sustainability of the SPI results [7].

Our interviewees have also stated that the success of the SPI projects is strongly related to software process ownership. According to them, not only management and SPI leaders should own the process, but also all the technical staff members. They should be responsible for the software process and its changes.

V. FINAL REMARKS

In this paper, we have presented thirty three success factors influencing the sustainability of SPI efforts. The factors are grouped into three categories: *organizational factors*, *implementation factors* and *social factors*. They were elicited in two independently conducted studies, the literature study and empirical study. The early results of this study were previously published in [1].

More than 70% of the identified SPI sustainability factors (24 out of 33) were commonly identified both via literature and empirical studies, even though they were conducted independently. Out of twenty seven success factors identified in the literature, only four were not mentioned by our interviewees. This represents a general approval of the SPI success factors identified in the literature, since the interviewees were not influenced by the results of the literature study. The factors that were not confirmed by the interviewees concerned two organizational factors dealing with the alignment of SPI with business goals and organizational policies and strategies (see Section IV.B.7 *SPI is aligned with business goals*, and Section IV.B.8 *SPI is aligned with organizational policies and strategies*) and one

implementation factor dealing with the definition of the SPI method (see Section IV.C.5 *SPI method is well defined*). The reason to why those factors were not mentioned by our interviewees could be that they concerned management and business level of the SPI project that was more common to larger organizations and less applicable for medium-sized organizations to which all our interviewees belonged to.

During the interviews, we have identified thirty SPI success factors, from which only six were not previously reported in the literature. Those factors concerned one organizational factor dealing with technical staff turnover (see Section IV.B.6 *The level of technical staff turnover is low*), three implementation factors dealing with process improvement and process standards (see Sections IV.C.8 *Process improvement effort is flexible*, Section IV.C.10 *Process standards are defined and enforced*, and Section IV.C.11 *SPI effort brings positive results*, and finally, two social factors dealing with the stakeholders' expectations from and encouragement towards SPI (see Section IV.D.3 *Stakeholders have realistic expectations*, and Section IV.D.6 *Stakeholders are being encouraged to support SPI*). Those factors represent lessons learned from SPI adaptation in the interviewed companies. Therefore, they may be context dependent and not necessarily applicable to other software organizations. Nevertheless, they clearly contribute to the SPI success and sustainability as stated in Section IV.

Despite the high overlap of the identified factors, the focus on the factors differed greatly among the sources used in this study. The literature sources mainly focused on conditions enabling the SPI efforts. The SPI factors that were recognized by the majority of the literature sources dealt with continuous management commitment and support, provision of resources and involvement of the technical staff in SPI (see Section IV.B.1 *Management continuously supports and commits to SPI process*, Section IV.B.2 *Resources are dedicated to SPI*, and Section IV.D.9 *Technical staff participates in SPI*). Those factors have the most devastating impact on the SPI project. Without them, the SPI project should not even be initiated. Their importance and influence were also confirmed during the interviews.

The factors that were mentioned by most of the interviewees focused on the social factors and the effects of SPI on the daily routines. Their concern mainly dealt with down to earth issues such as training in software process undergoing improvement, availability of the information on the SPI activities and results, and acceptance of the SPI activities by the technical staff (see Section IV.C.1 *Stakeholders are trained in software process*, Section IV.C.9 *Information about the SPI activities and its results is disseminated*, and Section IV.D.4 *Technical staff accepts SPI activities*). This may be explained with the fact that the majority of our interviewees have taken part in the SPI projects but they did not lead them. Therefore, our interviewees represented the developers' view from the perspective of how SPI affects their way of working and their daily routines.

The SPI sustainability success factors presented in this paper constitute the body of knowledge of the software engineering community as deduced in the current software engineering literature and in the industry. Even if they

have only been listed and described, they may already constitute a basis for providing insight into SPI efforts, for diagnosing the reasons of SPI decay or for confirming the prerequisites that are necessary for carrying out SPI.

We strongly believe that it is not enough to just define SPI process frameworks and/or models. The process frameworks/models should be supported by tools for evaluating the status of the SPI projects and identifying its faults. For this reason, we plan to continue working with the SPI sustainability success factors presented in this paper. Our goal is to create a basis for supplementing currently defined SPI frameworks and/or models with a checklist for SPI effort evaluation.

REFERENCES

- [1] N. Nikitina and M. Kajko-Mattsson, "Success Factors Leading to the Sustainability of Software Process Improvement Efforts," Proc. 6th International Conference on Software Engineering Advances, (ICSEA 2011). IEEE Press, 2011, pp. 581-588.
- [2] S. S. Chakravorty, "Where Process Improvement Projects Go wrong," Wall Street J., 2010.
- [3] N. Nikitina and M. Kajko-Mattsson, "Developer-Driven Big Bang Process Transition from Scrum to Kanban," Proc. International Conference on Software and Systems Process (ICSSP 2011). ACM, 2011, pp. 159-168.
- [4] D. Goldenson and D. Herbsleb, "After the Appraisal: A systematic Survey of Process Improvements, its Benefits, and Success Factors that influence Success," Technical report, SEI, 1995.
- [5] K. El Emam, P. Fusaro and B. Smith, "Success factors and barriers for software process improvement," 1999. In: C. Tully, R. Messnarz, (Eds), "Better Software Practice For Business Benefit: Principles and Experience," IEEE Computer Society Press, Silver Spring, MD.
- [6] D. Stelzer and W. Mellis, "Success Factors of Organizational Change in Software Process Improvements," J. Softw. Process Improve. Pract., 1998, pp. 227-250.
- [7] A. Rainer and T. Hall, "Key success factors for implementing software process improvement: a maturity-based analysis," J. Syst. Softw., vol. 62, 2002, pp. 71-84.
- [8] M. Niazi, D. Wilson and D. Zowdhi, "A framework for assisting the design of effective software process improvement implementation strategies," J. of Syst. and Softw., vol. 78, 2005, pp. 204-222.
- [9] M. Niazi, D. Wilson and D. Zowghi, "Critical Success Factors for Software Process Improvement Implementation: An Empirical Study," J. Softw. Process Improve. Pract., vol. 11, 2006, pp. 193-211.
- [10] T. Varkoi, "Management of Continuous Software Process Improvement," Proc. 2002 IEMC, 2002, pp. 334-337.
- [11] B. Curtis and M. Paulk, "Creating a software process improvement program," Butterworth-Heinemann Ltd, vol. 35: 6/7, 1993.
- [12] T. Dyba, "An instrument for measuring the key factors of success in Software Process Improvement," J. Emp. Softw. Eng., vol. 5, 2000, pp. 357-390.
- [13] T. Dyba, "An empirical investigation of the key factor for success in software process improvements," J. Trans. on Soft. Eng., vol 31: 5, 2005.
- [14] I. Aaen, J. Arent, L. Mathissen and O. Ngwenyama, "A conceptual MAP of Software Process Improvement," Scandinavian J. of Infor. Syst., vol. 13, 2001.
- [15] T. Galinac, "Empirical Evaluation of selected best practices in implementation of software process improvements," J. Infor. Soft. Techn., vol. 51, 2009, pp. 1351-1364.
- [16] A. Hudson, "Why Six Sigma Projects Fail & How to Prevent It", Groupiter Solutions Pty, [online resource]:

- http://www.grouputer.com/papers/why_six_sigma_projects_fail.pdf
- [17] M. Sulayman, C. Urquhart, E. Mendes and S. Seidel, "Software Process Improvement Success Factors for Small and Medium Web Companies: A Qualitative Study," *J. Inf. Softw. Technol.* 54, 2012, pp.479–500.
- [18] T. Hall, A. Rainer and N. Badoo, "Implementing Software Process Improvement: An Empirical Study," *J. Softw. Process Improve. Pract.*, vol. 7, 2002, pp. 3–15.
- [19] S. Beecham, T. Hall and A. Rainer, "Software Process Improvement Problems in Twelve Software Companies: An Empirical Analysis," *Proc. Emp. Softw. Eng.*, vol. 8, 2003, pp. 7–42.
- [20] D. Paulish and A. D. Carleton, "Case Studies of Software Process Improvement Measurement," *IEEE Computer*, vol. 27: 9, 1994, pp. 50 - 57.
- [21] M. Nazir, R. Ahmad and N. H. Hassan, "Resistance factors in the Implementation of Software Process Improvement Project in Malaysia," *J. of Comp. Science*, vol. 4: 3, 2008, pp. 211-219.
- [22] K. C. Dangle, P. Larsen, M. Shaw and M. V. Zelcovitz, "Software Process Improvement in Small Organizations: A case study," *IEEE Software*, vol.22:6, 2005, pp. 68-75.
- [23] G. Santos, M. Montoni, J. Vasconcellos, S. Figuerido, R. Cabral, et. al., "Implementing Software Process Improvements Initiatives in Small and Medium-Size Enterprises in Brazil," *Proc. QUATIC*, 2007, pp.187-196.
- [24] S. B. Basri and R. V. O'Connor, "Organizational Commitment Towards Software Process Improvement: An Irish Software VSEs Case Study," *Proc. ITSIM'10, IEEE*, 2010, pp.1456-1461.
- [25] M. Niazi, D. Wilson and D. Zowdhi, "Implementing software process improvement initiatives: An Empirical study," *Proc. PROFES 2006*, 2006, pp. 222 – 233.
- [26] M. Niazi, M. Ali Babar and J. M. Verner, "Software Process Improvement Barriers: A cross-cultural comparison," *J. Infor. and Softw. Techn.*, vol. 52: 11, 2010.
- [27] A. Sweeney and D. W. Bustard, "Software Process Improvement: making it happen in practice," *Soft. Qual. J.*, vol. 6, 1997.
- [28] P. Abrahamsson and N. Iivari, "Commitment in Software Process Improvement--In Search of the Process," *Proc. HICSS 2002*, 2002.
- [29] N. Nikitina and M. Kajko-Mattsson, "Historical Perspective of Two Process Transitions," *Proc. 2009 International Conference on Software Engineering Advances (ICSEA 2009)*, IEEE, 2009, pp. 289-298.
- [30] F. Ekdahl and S. Larsson, "Experience Report: Using Internal CMMI Appraisals to Institutionalize Software Development Performance Improvement," *Proc. EuroMicro 2006*, 2006.
- [31] I. Sommerville, *Software Engineering*, 8th ed. Harlow, England: Person Education Limited, 2007.
- [32] F. J. Pino, O. Pedreira, F. Garcia, M. R. Luaces and M. Piattini, "Using Scrum to guide the execution of software process improvement in small organizations," *J. Systems and Software*, vol. 83, 10, 2010, pp. 1662-1677.
- [33] N. Habra, S. Alexandre, J.-M. Desharnais, C. Y. Laporte and A. Renault, "Initiating software process improvement in very small enterprises: Experience with a light assessment tool," *J. Information and Software Technology*, vol. 50, 7–8, 2008, pp. 763-771
- [34] S. Zahran, "Software Process Improvement: Practical Guidelines for business success," Addison Wesley, 1998.
- [35] C. L. Yeakley and J. D. Fiebrich, "Collaborative Process Improvement: With Examples from the Software World," Wiley-IEEE Computer Society Press, 2007.
- [36] T. Dybå and T. Dingsoyr, N. B. Moe, "Process Improvement in Practice: A Handbook for IT Companies," Boston: Kluwer Academic Publishers, 2004.
- [37] M. C. Paulk, B. Curtis, M. B. Chrissis and C. V. Weber, "Capability Maturity Model for Software," v.1.1, Technical report, CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie Mellon University, 1993.
- [38] CMMI Product Team, "Capability Maturity Model Integration (CMMI)," v1.1. Pittsburgh, USA: Software Engineering Institute, Carnegie Mellon University, 2002.
- [39] A. Dorling, "SPICE: Software Process Improvement and Capability Determination," *J. Software Quality*, vol.2-4, 1993, Springer, Netherlands, pp.209-224.
- [40] M. Harry and R. Schroeder, "Six Sigma: The Breakthrough Management Strategy Revolutionizing the World's Top Corporations," New York: Currency, 2000.
- [41] J. Gremba and C. Myers, "The IDEALSM Model: A Practical Guide for Improvement. Bridge," 3 (1997): 19–23. [online resource]. <http://www.sei.cmu.edu/ideal/ideal.bridge.html>
- [42] ISixSigma, "Six Sigma DMAIC Roadmap," [online resource]: <http://www.isixsigma.com/dmaic/six-sigma-dmaic-roadmap>
- [43] S. Masters and C. Bothwell, "CMM Appraisal Framework," Version 1.0, (CMU/SEI-95-TR-001, ESC-TR-95-001). Pittsburgh, USA: Software Engineering Institute, Carnegie Mellon University, 1995.
- [44] M. Poppendieck, "Principles of Lean Thinking," Poppendieck.LLC, 2002.
- [45] M. Poppendieck and T. Poppendieck, "Lean Software Development: An Agile Toolkit," Addison-Wesley Professional, 2003.
- [46] K. Schwaber and M. Beedle, "Agile Software Development with SCRUM," Prentice Hall, 2001.
- [47] M. D. Myers, *Qualitative Research in Business and Management*, Sage publications, London, 2009.
- [48] N. Nikitina and M. Kajko-Mattsson, "Software Process Improvement Health Checklist," *Proc. EuroSPI 2012*, Springer, 2012, p. 85-96.
- [49] W. Trochim, *Research Methods: The Concise Knowledge Base*, Cornell University, 2005.

Evaluating Performance of Android Systems as a Platform for Augmented Reality Applications

Andrés L. Sarmiento, Margarita Amor, Emilio J. Padrón, Carlos V. Regueiro, Raquel Concheiro, and Pablo Quintía

Dept. Electronics and Systems

University of A Coruña

A Coruña, Spain

andreslopezsarmiento@gmail.com, margarita.amor@udc.es, emilioj@udc.es, cvazquez@udc.es

rconcheiro@udc.es, pquintia@udc.es

Abstract—Android is a free operating system for mobile devices that has become very popular these days. In this work we analyse the capabilities of an Android smartphone with the OpenGL ES API for the rendering of synthetic realistic images. The aim is to find out the facilities and the main limitations of the platform for the development of augmented reality games, studying the integration of synthetic information with the real environment using data from the camera and the positioning engine. Thus, our research covers mainly three fields: an analysis of the information provided by the camera, a study of the tracking and positioning capabilities of current Android devices and an outline of the rendering facilities usually found in these devices. The performance, in terms of frames per second and latency, has been tested in different smartphones, in addition to evaluate the reliability and efficiency of the sensors and the quality of rendering. In order to show all the results obtained from this study we have developed an augmented reality game trying to combine quality, performance and real-time interaction.

Keywords-Augmented reality; Android; Positioning sensors; Image processing; Realistic image synthesis

I. INTRODUCTION

This work extends a previous survey [1] about the current state and capabilities of augmented reality applications in Android smartphones, adding new tests and devices to the analysis. Smartphones have gathered functionalities and features of an increasingly number of different devices, from those used in a more professional environment (i.e., mobile devices, electronic agendas, GPS) to others with recreational aspects (such as cameras or video game consoles). Although this means an obvious saving of money and space, the major advantage of these new devices is the integration of all those capabilities in increasingly complex and innovative applications.

Most of the operating systems available for these devices have been developed ad hoc for each model, such as Apple's *iPhone OS* [2] or Samsung's *Bada* [3]. Android [4], however, has a very different origin since it is a multi-platform linux-based OS (rigorously, Android is really a software stack that includes an OS, middleware and a handful of applications) promoted by a group of companies. This open source and cross-platform nature, together with the growth

it has experienced over the past few years, giving access to a wide range of devices ranging from the low price terminals to the more expensive ones, made us adopt Android as the platform for this work.

Augmented reality (AR) [5] is one of the newest and most popular applications that have recently shown up within the sphere of smartphones. Most of the existing proposals may be classified at one of the following three groups: AR browsers; applications that allow us to move through a completely synthetic environment; and, lastly, applications that use the camera information to show virtual objects in the phone.

AR browsers are outdoor augmented reality applications that do geopositioning of virtual objects in the real world by using the orientation sensors and the GPS or a triangulation positioning system to determine the position where they must be placed [6], [7]. The information about the objects to be positioned is pre-computed and these applications do not demand a great accuracy in the positioning and orientation of the mobile device. AR browsers generally presents a good operation, showing real time information with an acceptable precision though with the typical limitations of any GPS or triangulation positioning systems. Generally speaking, the positioning accuracy is upper than 5 meters in optimal conditions (open space and good firmament visibility). However, the precision drops drastically in cities and, above all, inside buildings.

The second type of AR applications uses only the movement and orientation of the device to readjust the vision of a synthetically generated scene [8], [9]. In these applications all elements are generated in a virtual scene that is shown in the mobile screen. The device movements and its orientation in relation to the Earth's magnetic field and to the centre of the Earth (gravity) are used to establish or update the point of view in the scene shown in the screen. The image captured by the camera can also be shown, but it has not any influence in the applications as it is not really processed by the device.

Finally, some applications apply artificial vision techniques [10], [11]. This type of applications processes the

perceived image and uses that information to put the virtual models in the right place, usually through known tags to be able to interpret in a better way the perceived information. Obviously, this approach means higher computational requirements and a greater application complexity. As a consequence, there are really few AR applications in Android based on exploiting data obtained by the camera and most of them are basically technical demonstrations. On the other hand, this kind of applications can be found in other systems apart from mobile devices, such as desktop computers, mainly due to the usual high computing requirements of image processing algorithms.

To sum up, from a user point of view there are multiple applications using geopositioning of objects with different objectives, and several proposals already in the market to cover this topics. However, few Android applications use artificial vision to perceive and process the real world, and those have usually a poor performance. In our research, we focus on this last line of work since the best approach to integrate synthetic information with the immediate real-time data from the environment in a realistic scenario such as a dynamic and complex environment seems to be the exploitation of both the camera and the positioning sensors of these devices. Since Android is a brand new platform, analysing the viability of this kind of AR application is a necessary preliminary step. This analysis is complemented in this work with the development of a simple AR game for indoor environments as a demonstration of the possibilities of this approach.

The structure of the paper is as follows, Section II goes into the study of Android smartphones as an AR platform. We have divided our analysis in three big sections: firstly, a study of the possibilities for processing the information captured by the camera; next, a survey of the positioning and tracking capabilities of these smartphones in an indoor environment and, lastly, the possibilities for the real-time rendering of complex virtual models. A brief outline of all the aspects studied in the analysis is in the end of this section. Section III describes the AR game we have developed taking into account the results from our analysis, and Section IV shows the performance achieved with our proposal. Finally, the conclusions we have reached with this work are shown.

II. ANALYSIS OF THE CAPABILITIES OF AN ANDROID SMARTPHONE WITH OPENGL ES

In this section, an analysis of the capabilities of the Android platform in the context of AR is presented. Table I shows the main features of the devices used in our study. These devices are representative of the current smartphone market in the last couple of years.

A. Image capture and processing

The camera of a smartphone is of great importance for AR applications, since the synthetic images are usually rendered over real images obtained by the camera. If the image from the camera is just being displayed, Android efficiently add it to the rest of layers shown by the application. Otherwise, if the image is going to be processed, it is captured by the system and provided to the application as a vector of bytes in a default format previously set in the camera. Many cameras (such as the ones used in our analysis) only work with the YUV image format.

Once an image from the camera is obtained, any image processing technique may be applied on it. Since image processing is usually a high-cost computationally task, any operation has to be spawned in a different thread to the one running the application's GUI. Otherwise, non-responding lags are probably to be experienced in the application. Besides, it is also a good practice to code image processing tasks in native code (C, C++) and use the NDK to integrate it in the application [12]. This way, we can achieve an important improvement, up to 400%, in the velocity of execution.

In order to analyse the possibilities of image capture and processing at iterative rates we started studying the maximum frequency at which data can be obtained. This allow us to get the top level of performance that can be achieved in these devices. Thus, our first test just captures an input image and calls a naive processing image code that just computes the frame rate (fps, frames per second) with no additional computation (the input image is not processed at all). The result obtained with this simple test for a *Motorola Milestone* with Android v2.1 and a configuration of 10 fps as the maximum capture frequency was 8.8 fps, whereas a maximum of 9.3 fps was obtained when the maximum frequency was set to 30 fps. Obviously, these results are far from being satisfactory, since even without any image processing we are rounding the minimum frame rate acceptable for a fluid interaction.

To study the effect of a simple image processing on the performance, we have extended our test by adding the display on the screen of the images obtained by the camera. Since images are obtained from the camera in YUV format for portability issues and they must be in RGB to be displayed by Android, some computation is needed to get the conversion. Therefore, this test program just takes each image captured by the camera, recodes it from YUV to RGB and gets it displayed on the screen. Additionally, our test program can be configured to encode only a region of the image. The results of running our tests in the *Motorola Milestone* are depicted in Table II. The table shows the frame rate as a function of the size of the region to process and the highest frequency set in the camera. As can be observed, a top value of 5.15 fps has been obtained, that does not make

Table I: Technical data for the smartphones used in our tests.

Android	<i>Motorola Milestone</i> 2.1 Eclair	<i>GeeksPhone One</i> 2.2 Froyo	<i>Samsung Galaxy S</i> 2.2 Froyo	<i>HTC Sensation</i> 4.0 ICS	<i>Samsung Galaxy S2</i> 2.3 Gingerbread / 4.0 ICS
CPU	ARM Cortex A8 550 MHz	ARM11 528 MHz	Samsung Hummingbird 1 GHz	Qualcomm Scorpion dual-core 1.2 GHz	ARM Cortex A9 dual-core 1.2 GHz
GPU	PowerVR SGX 530	built-in	PowerVR SGX 540	Qualcomm Adreno 220	ARM Mali-400 MP
RAM	256 MB	256 MB	512 MB	768 MB	1 GB
Display	3.7" 854x480	3.2" 400x240	4" 800x480	4.3" 960x540	4.3" 800x480
GPS	✓	✓	✓	✓	✓
Acceler.	✓	✓	✓	✓	✓
Compass	✓	✗	✓	✓	✓
Camera	✓	✓	✓	✓	✓

possible to keep a fluid stream of images on the screen. Furthermore, we have observed a delay of about one second in what is being displayed. A further analysis of this delay is presented at the end of this section.

These results show that the configuration with 10 fps as the maximum frequency obtains the best results; probably because with this frequency the application is not saturated with images it is not able to process. Even though there is a substantial improvement in the capture by reducing the image size (25% fps with 1/4 of the size), this results in less than 5 fps with the max set to 30 fps.

In a next step we have studied the performance of image processing, so a simple colour segmentation of pixels is carried out. Since all the pixels in the image were already being processed by the image recoding process in the previous test, adding colour segmentation only needs a few additional lines of code, so execution times remains almost the same, as we have tested experimentally. Other tests adding different image processing algorithms were carried out and similar execution times were obtained.

The obvious conclusion coming from the results of our tests is that the image processing velocity is really low in Android v2.1 and previous versions, obtaining a slow response even after implementing optimisations such as using NDK and running the processing in a different thread. The main reason for this performance seems to be in the process the system follows for each image captured by the camera, allocating memory, saving a copy of the image, calling the function to process it and, finally, removing the reference to the allocated memory [13]. This whole process entails a completely inefficient memory management, that is made still more acute by the high cost of garbage collection in Android, between 100 and 300 milliseconds. Not reusing the memory assigned to each image results in a frequent invocation of the garbage collector, burdening the performance.

This important issue with memory management was solved in Android v2.2, that included other significant improvements as well, such as a *Just in Time* compiler. Regarding image processing, the API was also enhanced with new methods that work with a buffer passed as a

Table II: Image capture, decoding and visualisation on *Motorola Milestone* with Android v2.1.

Image size	Max. FPS	
	<i>Milestone-30</i>	<i>Milestone-10</i>
560×320	3.25	3.90
280×320	3.90	4.45
280×160	4.50	4.95
140×160	4.60	5.10
15×15	4.65	5.15

Table III: Image capture, decoding and visualisation in devices with Android v2.2.

<i>GeeksPhone</i>		<i>Galaxy S</i>	
Size	FPS	Size	FPS
400×240	3.90	800×480	5.70
200×240	4.50	400×480	7.10
200×120	5.00	400×240	8.00
100×120	5.50	200×240	8.75
15×15	5.80	15×15	9.20

parameter, removing the memory allocation and removal for each image to process.

We have analysed the improvements in Android v2.2 by running the same tests in two of our devices with this version of the OS. Table III shows the results obtained with Android v2.2 for the simple capture and recoding test previously outlined in Table II, in this case considering only the configuration of 10 fps as the maximum frequency, since it provides the best results and the 30 fps configuration does not add relevant information to the analysis. As can be observed, there is a performance increase of 50%, from 3.90 up to 5.70, and taking into account a 50% increase in the image size as well. The improvement is even more appreciable looking at the visualisation delay, that has been reduced from around 1 second to 0.5 seconds.

Table IV depicts the results achieved by the new Android v4.0 running on two current smartphones: HTC Sensation and Samsung Galaxy S2. As can be observed, the performance has significantly improved, above all for the 30 fps configuration, that now even achieves the best results in some of the cases. The frame rates shown in this table

Table IV: Image capture, decoding and visualisation on *HTC Sensation* and *Galaxy S2* with Android v4.0.

Image size	Max. FPS			
	<i>Sensation-30</i>	<i>Sensation-10</i>	<i>Galaxy S2-30</i>	<i>Galaxy S2-10</i>
640×480	9.38	9.62	11.89	11.59
320×480	10.36	10.50	11.93	11.87
320×240	12.20	12.20	12.08	12.04
160×240	12.64	12.71	13.26	12.06
160×120	13.28	13.05	14.87	12.63

Table V: Image capture, decoding and visualisation in a high load scenario on *HTC Sensation* and *Galaxy S2* with Android v4.0.

Image size	Max. FPS			
	<i>Sensation-30</i>	<i>Sensation-10</i>	<i>Galaxy S2-30</i>	<i>Galaxy S2-10</i>
640×480	7.60	7.70	8.60	8.73
320×480	8.00	8.10	9.41	9.23
320×240	8.40	8.50	12.03	9.60
160×240	8.90	8.70	13.03	9.76
160×120	9.10	8.90	14.42	10.01

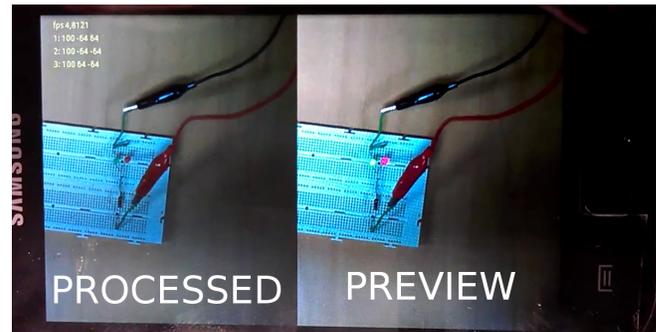
were obtained while keeping a small amount of background workload in the smartphone, as in the previous tests. In order to check how the background processes running in the smartphone influence the performance of the image capture and process task, we have repeated the test in the Android v4.0 devices while a great bunch of usual applications were being executed in background (IM, e-mail, alerts...). The results in such a scenario are shown in Table V, and an important drop in performance can be observed compared to Table IV, above all in the *HTC Sensation*, even though both two devices are dual core.

Lastly, all our tests analysing the image capture and processing in Android have revealed an important delay in the capture of the input data. This delay was extremely high in the devices with Android v2.1, about 1 second, and has been reduced in the last versions. Table VI shows the results of a simple experiment to measure this delay in three different devices, *Samsung Galaxy S*, with Android v2.2, and *HTC Sensation* and *Samsung Galaxy S2*, with Android v4.0. Our experiment involved the measurement of the response time of a simple quantifiable event, the off/on switching of a LED, that allow us to know the delay in the image capture for each device (see Figure 1). As can be observed, the delay in the *Samsung Galaxy S2* is about half the time it is in the other two devices, even though one of them, *HTC Sensation*, is using the same version of Android. A video with this experiment is provided as additional material with this paper.

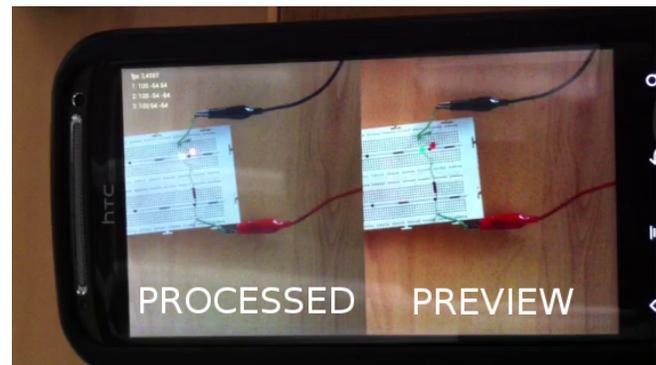
Summing up, although the improvements introduced with Android v2.2 and next versions make us optimistic about future revisions, above all in combination with more powerful hardware such as the recent multi core processors,

Table VI: Delay between image capture and processing on *Galaxy S* with Android v2.2 and *HTC Sensation* and *Galaxy S2* with Android v4.0.

Delay (s)	<i>Galaxy S</i>	<i>Sensation</i>	<i>Galaxy S2</i>
Average	0.454	0.414	0.246
Std. deviation	0.024	0.041	0.060



(a)



(b)

Figure 1: Delay time measurements in (a) *Samsung Galaxy S* and (b) *HTC Sensation*.

the current situation does not allow real time applications entirely based on the processing of images from the camera. Thus, an efficiency analysis of the real world around us makes necessary the use of data from other sources, e.g., positioning sensors.

B. Device positioning and orientation

In this subsection we outline the main positioning and tracking sensors included in most Android smartphones: accelerometer, compass and GPS. In order to check their performance, some test were executed on our *Milestone* phone, similar results were obtained in the rest of devices.

An accelerometer measures the proper acceleration of itself, i.e., a change in velocity, that involves a change in position. Mathematically velocity is the integral of acceleration, and position is the integral of velocity. Smartphones have usually three accelerometers, one for each spatial axis.

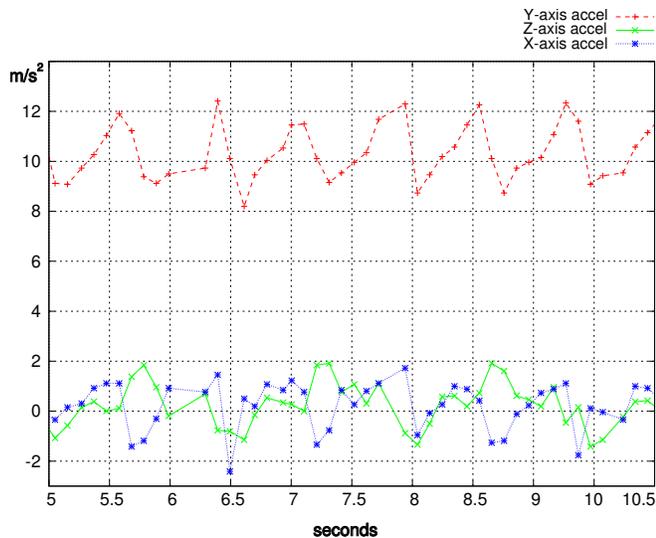


Figure 2: Values obtained by the accelerometers of a *Motorola Milestone* during a user's walk.

Theoretically, the position of a smartphone could be guessed from data provided by these sensors. Data are presented as a vector with the values measured for each of the three axis in SI units (m/s^2). In practice, however, the measures are not very accurate due to the presence of gravitational and centripetal forces [14]. Thus, a mobile phone left to stand on a table presents a downward acceleration of about $9.82m/s^2$, the gravitational acceleration. It will be necessary a gravitational free fall toward the centre of the Earth to measure a value of zero in the three axes. Furthermore, the double integral that has to be solved for obtaining location from the acceleration value also increases the measurement error. Anyway, these sensors are handy for knowing the device's position relative to the floor with simple trigonometric calculations.

To test these devices a small application that simply takes the received measurements and save them in a file has been developed. Figure 2 depicts the values received while a user is walking along the Z axis with the mobile vertical to the floor (axis Y is perpendicular to the floor and axis X is on the side). As can be seen, there is a regular pattern of about a footstep per second crests in axis Y (continuous changes of about $4m/s^2$): acceleration progressively increases each time the user raises his foot to start a new step, and it falls when the foot reaches the floor, before starting a new step and so on. The lateral movement enclosed to each footstep can also be observed, but more complex movements would be hard to recognise, hence the difficulty of computing displacements using acceleration values. Broadly speaking, the accelerometers we have tested measure a lot of noise and therefore don't seem reliable enough for a real time application.

A digital compass or magnetometer is a device that

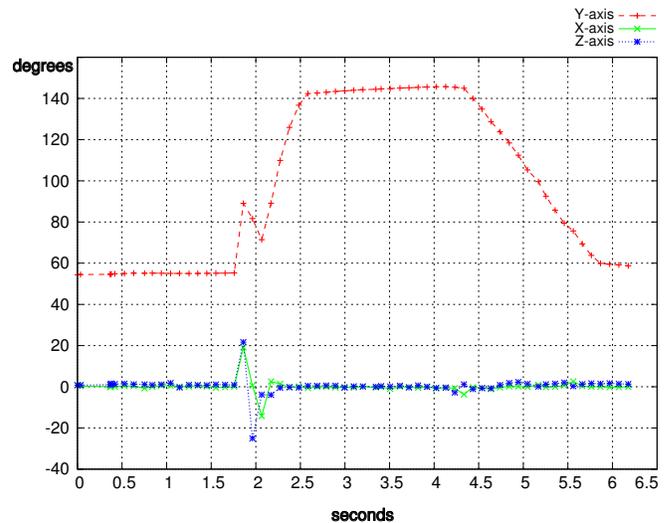


Figure 3: Values obtained by the compasses of a *Motorola Milestone* with a 90° turn.

measures the strength and direction of the magnetic fields in its environment. Due to the magnetic field of the Earth, a compass is usually employed as an orientation device since it points out the Earth's magnetic north. A smartphone usually incorporates a chip that integrates three compasses arranged to detect magnetic fields in the three spatial axes [15]. Thus, location can be obtained independently of the position of the device. Data are presented again as a three-component vector, with the rotation value for each axis. The first component of the vector is the rotation measurement usually employed, i.e., rotation with regard to z axis. This value is 0° when the device is oriented towards north direction and the value increases clockwise up to 360° , north again. The other two components, rotation regarding the other axes, are 0° when the device is lying face up. Figure 3 shows the results obtained by a test consisting of making an abrupt 90° turn, almost instantaneous, just before returning to the initial position by means of a slighter turn, during about 3 seconds. As can be observed in the figure, the compass is too slow in measuring the new position after the first sudden movement, what introduces wrong values during a short period. However, it behaves really well in the presence of slight movements, with accurate values and very little noise.

Therefore, to track the direction in which a smartphone moves with Android is recommended to take together data from the accelerometers and the compasses. By previously setting a default position for the device when the application starts we get enough accuracy, since measures of smooth changes in the local environment are quite precise.

GPS is a space-based global navigation satellite system that provides reliable location through an infrastructure comprising a network of satellites. This system can be used

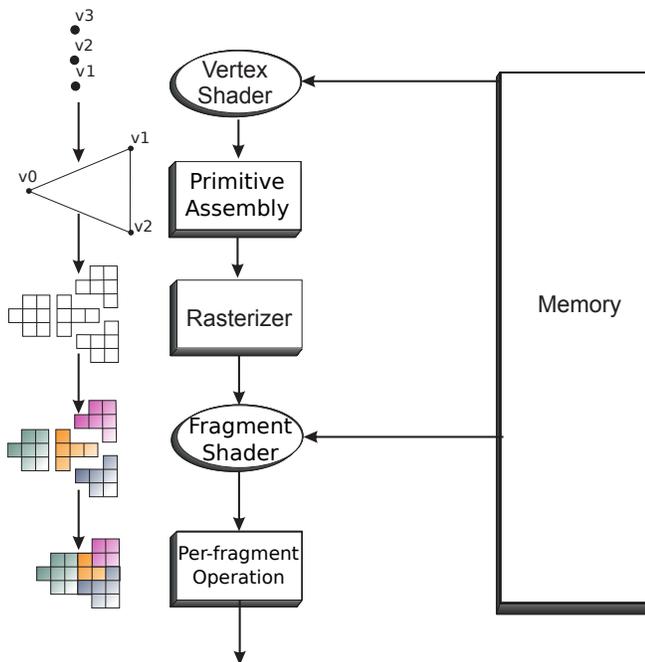


Figure 4: OpenGL ES 2.0 pipeline.

all around the world whenever there is an enough number of visible satellites. Otherwise, less accurate measurements are obtained or the device can even get out of network coverage, usual problem in the indoor locations. The values obtained by a GPS device points out its current position in the globe with a few meters of precision, about 10 meters outdoor. Besides, it does not provide reliable information about the direction or inclination of the device and data is obtained with a delay of about 1 and 2 seconds. All this makes difficult to realistically locate and move synthetic objects that are close to the device.

Nowadays, an alternative method to GPS is network-based tracking by using the service provider's network infrastructure to identify the location of the device. Although this technique has less accuracy than GPS, it has the advantages of a shorter initialisation time and an important reduction in power consumption, since only the telephone signal is used. Additionally, it can provide better results in indoor environments than GPS. Anyway, both the two methods are compatible as they are not mutually exclusive.

C. Android and synthetic graphics

OpenGL ES [16] is the API for producing computer graphics in Android. It is a subset of the standard OpenGL designed for embedded devices, so it removes some redundant API calls and simplifies other elements to make it run efficiently on the low-power GPUs of handheld devices. Figure 4 depicts the rendering pipeline in OpenGL ES 2.0, with programmable hardware. Our codes are based on OpenGL ES 1.1, with the same pipeline but with configurable fixed

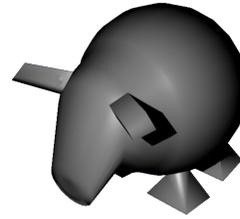


Figure 5: Test model for OpenGL ES.

Table VII: Performance (fps) of OpenGL ES in Android v2.1 and v2.2.

Points	GeeksPhone				Milestone				Galaxy S			
	C1	C2	C3	C4	C1	C2	C3	C4	C1	C2	C3	C4
3 K	35	35	35	33	30	30	30	30	55	55	55	55
9 K	18	19	19	15	30	29	29	28	55	55	55	55
15 K	12	10	10	10	29	26	26	25	55	55	55	55
30 K	8	-	-	-	25	22	22	19	55	55	55	55
75 K	-	-	-	-	18	15	15	12	55	53	53	50
100 K	-	-	-	-	-	-	-	-	55	44	44	41

function hardware instead of programmable shaders. A set of tests were carried out on the devices presented in Table I to analyse the performance of graphic synthesis in Android.

The first test focused on measuring performance as the number of primitives to render increases. The experimental results obtained for a scene with the model of Figure 5 replicated multiple times are shown in the column C1 of Table VII, in this case using the smartphones with Android v2.x: *GeeksPhone*, *Milestone* and *Galaxy S*. In view of these results, it is clear that the performance gets worse as the number of polygons increases except for *Galaxy S*, device in which we perceive a serious performance loss starting from 300K points. Non relevant values were not included in the table.

Column C2 of Table VII shows the results after adding a texture to the model of Figure 5. This definitely improves the visual aspect of the virtual objects with a minimum loss of efficiency, up to a 17% for a model of 75000 points in our *Milestone*. Column C3 depicts the results when including transparency effects. This hardly has influence on performance comparing to the synthesis with textures. In column C4 the results are obtained after applying illumination to the models. The performance decreases now a 24% in *Milestone* for a scene with 30K points. Obviously, this loss of performance is due to the additional computation required to get the colour of each pixel in the scene. Furthermore, it is necessary to define the light sources in the scene, setting its position, type, colour and intensity, in addition to provide each vertex of the models with a normal vector. As can be observed, the fall of performance in *Galaxy S* is only noticeable for models with a certain complexity (100K points).

Table VIII: Performance (fps) of OpenGL ES in Android v4.0 (also Android v2.3 in Galaxy S2).

Points	Sensation				Galaxy S2							
	C1	C2	C3	C4	C1		C2		C3		C4	
					2.3	4.0	2.3	4.0	2.3	4.0	2.3	4.0
30 K	60	60	60	60	60	60	60	60	60	60	60	60
75 K	56	55	55	54	60	60	60	60	60	60	46	52
100 K	54	53	53	50	59	59	60	60	60	57	33	37
150 K	49	48	49	24	56	58	60	60	60	60	23	26
180 K	43	40	37	18	50	55	49	50	52	56	20	22
210 K	37	32	33	14	45	48	43	44	41	46	17	19
240 K	31	26	26	12	40	43	38	39	37	39	14	17
270 K	25	21	23	10	36	38	32	35	35	37	13	15
300 K	21	17	16	9	31	35	30	32	31	33	11	13

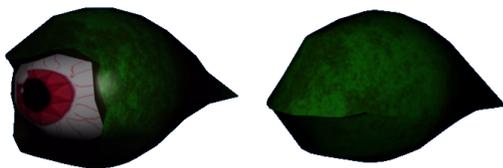


Figure 6: Morphing animation: starting state on the left and final state on the right.

Table VIII has the results for the smartphones with Android v4.0 used in our work: *Sensation* and *Galaxy S2*. *Sensation* keeps a good performance up to 180-210 K points, though the frame rate with illumination (column D) dramatically drops for more than 150 K points. *Galaxy S2* obtains very similar results to *Galaxy S*, keeping a good performance even with 300 K points. Again, working with illuminated models makes the performance drop, surprisingly resulting in poorer frame rates than *Sensation* for 75-100K points. The table also has the results for Android v2.3 in the *Galaxy S2* model, what shows the noticeable improvement introduced with Android V4.0, above all when the number of primitives to be rendered increases.

As regards animation, among all the different methods we have analysed the inclusion of morphing [17]. This technique gets a smooth transition between two models, using interpolation to compute the intermediate versions of the models. Since a new position for each point in the model has to be calculated for each frame, this kind of methods have a high computational cost. The model in Figure 6 (around 800 points and 300 polygons) has been used to test the performance of this kind of animation together with the application of textures and illumination in our target devices. The frame rates obtained for different scenes with this model on the smartphones with Android v2.x are shown in Table IX (only the most interesting results were measured). It can be observed that performance falls off dramatically except for low-complexity scenes (8K in the

Table IX: Frame rate comparison of static (S) and animated (A) models in the scene with Android v2.1 and v2.2.

Points	<i>GeeksPhone</i>		<i>Milestone</i>		<i>Galaxy S</i>	
	S	A	S	A	S	A
800	40	21	30	30	55	55
1.6 K	32	14	30	25	55	55
2.4 K	27	10	30	18	55	55
4 K	21	6	30	10	55	51
8 K	-	-	27	5	55	29
12 K	-	-	-	-	55	20
16 K	-	-	-	-	55	15

Table X: Frame rate comparison of static (S) and animated (A) models in the scene with Android v4.0 (also Android v2.3 in Galaxy S2).

Points	<i>Sensation</i>		<i>Galaxy S2</i>			
	S	A	S		A	
			2.3	4.0	2.3	4.0
30 K	60	60	60	60	60	60
75 K	60	53	60	60	51	60
100 K	59	39	60	60	36	60
150 K	57	28	60	60	24	46
180 K	55	25	55	60	19	39
210 K	54	21	47	60	17	33
240 K	54	19	42	60	14	30
270 K	47	17	41	53	12	27
300 K	44	14	37	48	11	23

case of *Galaxy S*). These results are greatly improved in the new devices with Android v4.0, as shown in Table X. Specifically, the rendering of animated models has the most remarkable improvement, above all in the *Galaxy S2*, that exhibits a great performance up to 150 K points and keeps good frame rates with 240 K points in the scene. Again, the table allow us to compare the results accomplished by Android v2.3 and v4.0 when running on the same hardware (*Galaxy S2*). The improvements introduced in Android v4.0 lead to an increase in performance when the number of primitives to be rendered is greater than 150K.

D. Discussion

An important deficiency in the image processing capabilities of the platform has arisen, mainly in terms of image capture latency (a minimum of 0.246 seconds in high-end smartphones). The main augmented reality applications of other platforms use the information obtained after a complex analysis of the images captured by the camera as the main source of information for positioning the synthetic objects in the scene. In view of the results of our analysis, this kind of applications are currently not possible at all in the Android devices we have tested.

On the other hand, multiple conclusions can be extracted from the analysis carried out using the Android positioning sensors. First of all, regarding the use of the built-in locating and tracking sensors, the accelerometers and the compass provide results relatively reliable with no important errors.



Figure 7: Three-dimensional models.

However, GPS gives an excessive error in the measure to be used in the kind of AR indoor application we propose in this work.

Lastly, we have detected restrictions in size and complexity of the models to be rendered. From the results we can deduce that the graphic hardware is powerful enough to render non-excessively complex models with textures and illumination. Therefore, in the game we propose in the next section as an example of AR application, all the render capabilities we have analysed have been applied, but limiting the complexity of the model in order to get real time rendering.

III. AN AR GAME IN AN ANDROID PLATFORM

To exploit the different aspects we have studied in our analysis we have developed a simple AR game. In this game each real-time image obtained by the camera is analysed and it determines the apparition of 'enemies' that the user/player must hunt down. Thus, we have implemented a simple system of events based on object colours and the different enemies are drawn when a certain event is triggered. Therefore, in accordance with this game idea, the main requirement is to render virtual elements (the 'enemies') on the live image stream from the camera. These synthetic characters have to look as if they really were in the real world so they must behave properly with camera movements.

There are 4 different enemies in the game, each one of them with specific reactions and movements: *BatGhost* (Figure 7a), has been designed as an example of animation by parts, with its wings moving independently to provide a sense of flapping, *HulkGhost* (Figure 7b) with its blinking eye is an example of animation using morphing techniques, *EmGhost* (Figure 7c) was designed to have an enemy with

Table XI: Frame rates of the AR game.

Test	<i>Milestone</i>		<i>GeeksPhone</i>		<i>Galaxy S</i>		
	ImPr	Syn	ImPr	Syn	ImPr	Syn	
Static		3.25	15	2.75	8	4.10	35
	ImPr deact.		30		21		44
Anim.		2.75	8	2.50	3	3.60	23
	ImPr deact.		28		17		41

bouncing capabilities, that could jump over the player, and *SuperGhost* (Figure 7d), that moves around the player while approaching to him. Whenever an enemy is hunted, the player earns points and extra shoots as a reward. Otherwise, if the player is hit by the enemy a life is lost. When the player loses its last life the game ends. Figure 8 shows some screenshots of the game: an enemy appears when the associated colour is detected through the camera (Figure 8a), the settings menu of the application (Figure 8b) and the colour calibration process (Figure 8c). This calibration is indispensable to get a right colour-based event triggering with different devices.

When it comes to rendering the different elements through OpenGL ES calls, the operating system itself executes these calls in a different thread, allowing a decoupled execution [18]. Furthermore, the reuse of memory is a constant issue in our implementation, preventing the number of memory allocations as far as possible.

IV. EXPERIMENTAL RESULTS

This section presents the performance achieved by our AR application. The resulting frame rates are shown in Table XI. The different columns show the frames per second for image processing (ImPr) and image synthesis (Syn) in each device. The results in rows 2 and 4 (ImPr deact.) are obtained by deactivating the image processing task once an event is triggered, as described below.

In *Motorola Milestone* the image processing rate ranges from 3.25 fps with no visible enemy to 2.75 fps when an animated (morphing) enemy appears. Besides, the image synthesis rate falls down from 15 fps to 8 fps with only an animated model in the scene.

The performance is slightly worse in *GeeksPhone One*, with a peak of 2.75 fps for image processing. As can be seen, the main performance loss is mostly noticeable in the graphic synthesis. While the stream of images obtained from the camera is being processed, the performance values of the graphic synthesis are lower than the ones for *Motorola Milestone* in about 50%.

In the case of *Galaxy S* we have obtained better results, with a rate of image processing ranging from 3.6 fps to 4.1 fps along with a rate of synthesis of 35 fps for static models and 23 fps for animated, aspect in which the improvement is more appreciable.

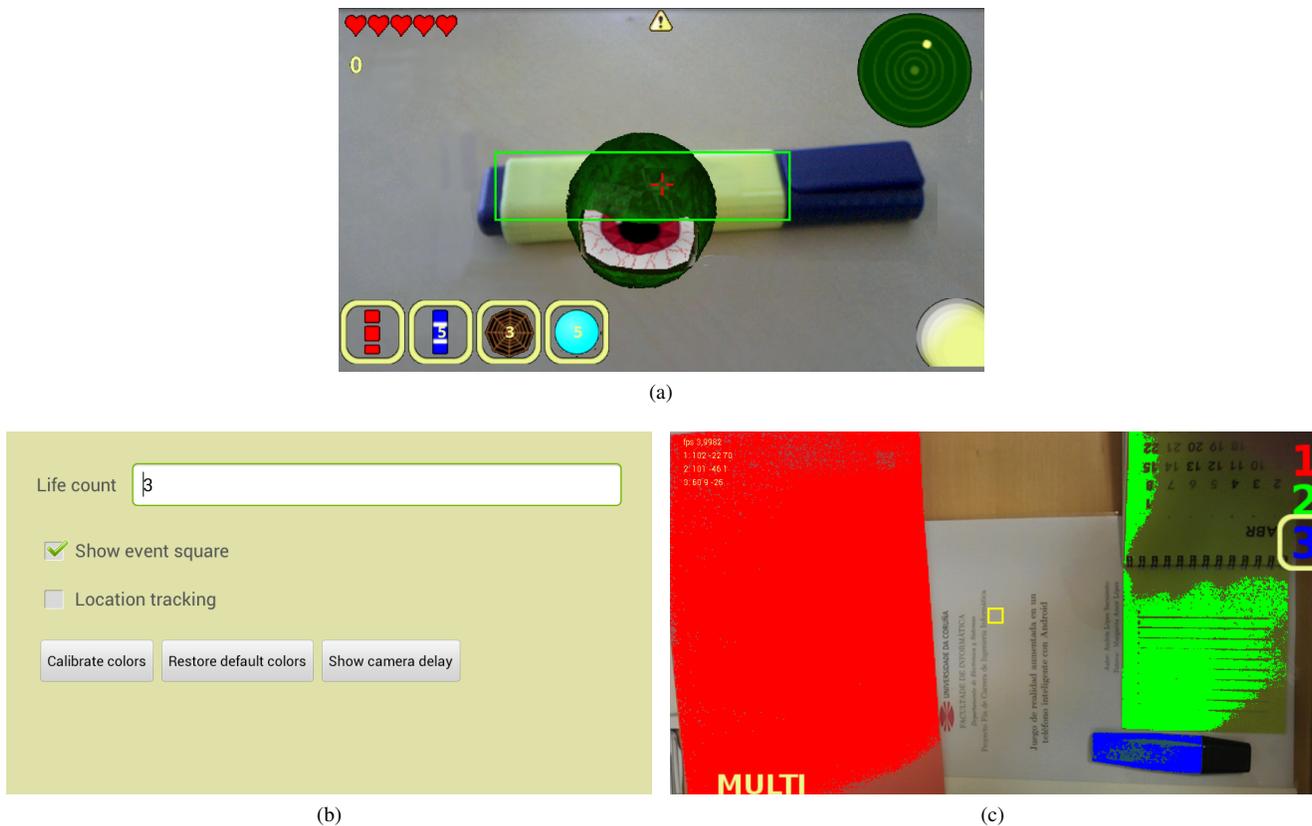


Figure 8: Game screenshots: (a) Event detection and triggering (b) Settings menu (c) Color calibration.

On the other hand, the performance loss in the processing of the image has increased the delay in obtaining new images from the camera, reaching now about 1 second in our application.

As commented, once an enemy is discovered it does not keep still, it moves around the environment. To increase the frame rate and achieve a good response and fluid feeling we have stopped the image processing task while the enemy remains active in the screen. This restricts the appearance of multiple simultaneous enemies, but allow us to get an outstanding improvement in the rendering, reaching about 30 fps in *Milestone*, 21 fps in *GeeksPhone* and 44 fps in *Galaxy S*, a performance high enough to achieve an acceptable fluidity in an AR game.

V. CONCLUSIONS

In this work we present a study of the capabilities of current smartphones in the context of AR applications. Thus, to test the feasibility of this kind of applications we start checking the main constraints in the obtaining of data from the device's camera. The maximum frame rate we can obtained is less than 15 fps in the best cases. Including additional processing, such as colour segmentation, does not have an appreciable impact in performance. The main limitation is the latency in the image capture, near to 0.25

seconds in the best case. However, looking at the evolution of this delay, from about 1 second in the oldest analysed devices, rounding 0.5 seconds in the mid-range phones and about 0.25 in the *Galaxy S2* with Android v4.0, it seems that this drawback will be solved in a near future. Besides, using native code and the NDK seems essential to achieve a good performance, as the existing software layers are probably introducing the main performance issues (drawback of being a multiplatform OS).

Another point in our study has been to analyse the locating and tracking features of these devices. From our tests we have concluded that to obtain the device orientation is relatively simple and reliable. Nevertheless, to guess the device displacement is really complicated. Calculating it using the values obtained by the accelerometers is not very reliable, due to the numerical errors in the computation of the double integration. Additionally, geolocation systems have a margin of error too high for our requirements, about 10 meters.

With regard to the rendering of synthetic images with the OpenGL ES library, we have tested the inclusion of textures, illumination and transparencies. The performance achieved in scenes with up to 15K points has been acceptable for a mid-range smartphone as *Motorola Milestone*. Adding models with morphing animation means a loss higher than

20% each time the number of points is doubled.

As a proof of concept, to show the possibilities within the AR field of the different smartphones we have analysed, an interactive AR video game has been implemented. The performance we have achieved in this application is 3.25 images obtained through the camera per second and 28 fps in the synthesis of graphics in a mid-high end smartphone as *Motorola Milestone*. The results are better in a more powerful device as *Samsung Galaxy S*, 4.1 processed images per second and 35 fps, and appreciably worse in a low-end device as *GeeksPhone One*, 2.75 processed images per second and only 8 fps.

Additionally, it should be pointed out the great influence that the presence of other running tasks (background and foreground, e.g., a chat) have in the performance of an AR application. In this sense, a further analysis of the task scheduling on Android would be interesting.

As future work, we plan to propose a set of benchmarks in order to identify graphics processor bottlenecks. This proposal aims to guide programmers to identify the benefits of potential optimisations. These benchmarks could be a useful tool to make design decisions in architecture improvements.

Specifically, this proposal is aimed at designing, implementing and testing a set of benchmarks to analyse the rendering capabilities of Bézier surfaces on an Android smartphone with the OpenGL ES API. We expect to describe several hand-tuned Bézier Surfaces rendering in real-time implementation on Android systems, identifying key graphics processor performance limitations, enhancements and tuning opportunities.

ACKNOWLEDGEMENTS

This work has been economically supported by Ministry of Education and Science of Spain under the contracts MEC TIN 2010-16735 and TIN 2009-07737 and also by the Galician Government under the contracts 'Consolidation of Competitive Research Groups, Xunta de Galicia ref. 2010/6', 08TIC001206PR, INCITE08PXIB105161PR, INCITE08PXIB262202PR, and CN2012/211, partially supported by FEDER funds.

REFERENCES

- [1] A. L. Sarmiento, M. Amor, C. V. Regueiro, and E. J. Padrn, "An analysis of android smartphones as a platform for augmented reality games," in *Proceedings of the Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2011)*, November 2011, pp. 71–76.
- [2] A. Inc., "iOS Dev Center," <http://developer.apple.com/devcenter/ios/index.action>, last access: 20/12/2012.
- [3] S. E. Co., "Bada developers," <http://developer.bada.com/>, last access: 20/12/2012.
- [4] M. Gargenta, *Learning Android*. O'Reilly, 2011.
- [5] D. Wagner and D. Schmalstieg, "Making augmented reality practical on mobile phones," *IEEE Computer Graphics and Applications*, vol. 29, no. 3, pp. 12–15, 2009.
- [6] Layar, "Layar reality browser," <http://www.layar.com>, last access: 20/12/2012.
- [7] T. Langlotz, C. Degendorfer, A. Mulloni, G. Schall, G. Reitmayr, and D. Schmalstieg, "Robust detection and tracking of annotations for outdoor augmented reality browsing," *Computer & Graphics*, vol. 35, no. 4, pp. 831–840, 2011.
- [8] MADfirm, "Sky siege," <http://madfirm.com>, last access: 20/12/2012.
- [9] Quest-Com, "Droidshooting," <http://www.quest-com.co.jp>, last access: 20/12/2012.
- [10] H. Kato, "Artoolkit," <http://www.hitl.washington.edu/artoolkit>, Android port by A. Igarashi (2010), last access: 20/12/2012.
- [11] N. SL., "Invizimals," <http://www.invizimals.com>, last access: 20/12/2012.
- [12] S. Lee and J. W. Jeon, "Evaluating performance of android platform using native c for embedded systems," in *Proceedings of the International Conference on Control, Automation and Systems*, 2010, pp. 1160–1163.
- [13] "Android Google Code. Issue 2794: Camera preview callback memory issue," <http://code.google.com/p/android/issues/detail?id=2794>, last access: 20/12/2012.
- [14] R. Meike, "Location, location, location (accelerometer)," https://blogs.oracle.com/roger/entry/location_location_location_accelerometer, last access: 20/12/2012.
- [15] "Asahi Kasei Microdevices. 3-axis electronic compass," <http://www.asahi-kasei.co.jp/akm/en/index.html>, last access: 20/12/2012.
- [16] D. Astle and D. Durnil, *OpenGL ES Game Development*. Thomson Course Technology, 2004.
- [17] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*. A. K. Peters, 2008.
- [18] C. Pruet, "Writing real time games for android," <http://www.google.com/events/io/2009/sessions/WritingRealTimeGamesAndroid.html>, May 2009, last access: 20/12/2012.

Debugging Ubiquitous Computing Applications With the Interaction Analyzer

Nam Nguyen, Leonard Kleinrock, and Peter Reiher

Computer Science Department, UCLA

Los Angeles, CA, USA

songuku@cs.ucla.edu, lk@cs.ucla.edu, reiher@cs.ucla.edu

Abstract—Ubiquitous computing applications are frequently long-running and highly distributed, leading to bugs that only become apparent far from and long after their original points of origin. Such bugs are difficult to find. This paper describes the Interaction Analyzer, a debugging tool for ubiquitous computing applications that addresses this problem. The Interaction Analyzer uses protocol definitions and histories of executions that displayed bad behavior to assist developers in quickly finding the original root cause of a bug. We discuss characteristics of ubiquitous computing applications that can complicate debugging. We describe the architecture of the Interaction Analyzer and the methods it uses to rapidly narrow in on bugs. We also report overheads associated with the tool, simulation studies of its ability to find bugs rapidly, and case studies of its use in finding bugs in real ubiquitous computing applications.

Keywords—ubiquitous computing; distributed debugging; ubiquitous applications

I. INTRODUCTION

Ubiquitous and pervasive computing systems are often complex systems consisting of many different objects, components and agents, interacting in complicated and unpredictable ways. The real world frequently intrudes into pervasive systems, adding to their unpredictability. As a result, such systems can frequently display unexpected, and often erroneous, behaviors. The size and complexity of the systems and their interactions make it difficult for developers to determine why these unexpected behaviors occurred, which in turn makes it difficult to fix the problems [1][2][3][4].

We built a system called the Interaction Analyzer to help developers of complex ubiquitous computing systems understand their systems' behaviors and find and fix bugs [1]. The Interaction Analyzer gathers data from test runs of an application. When unexpected behavior occurs, it uses the data from that run and information provided during system development to guide developers to the root cause of errors. The Interaction Analyzer carefully selects events in the execution of an application and recommends that the human developers more carefully examine them. In real cases, the Interaction Analyzer has guided ubiquitous application developers to the root cause of system bugs while only requiring them to investigate a handful of events. In one case, the Interaction Analyzer helped developers find a race condition that they were previously unable to track down; the entire debugging process took less than five minutes, while previously developers had

spent several days unsuccessfully tracking the bug using more traditional debugging techniques.

In this paper, we describe how the Interaction Analyzer works and give both simulation results of its efficiency in tracking bugs and cases where it found real bugs in a real ubiquitous application. Section II describes the Panoply system, for which the Interaction Analyzer was built, and introduces the example ubiquitous application. Section III describes the Interaction Analyzer's basic design and architecture. Section IV provides simulation results and real case studies; this section also includes basic overhead costs for the Interaction Analyzer. Section V discusses related work and Section VI presents our conclusions.

II. PANOPLY AND THE SMART PARTY

The Interaction Analyzer was designed to be a general-purpose system usable in many ubiquitous computing contexts. However, since we wished to demonstrate its use in a real environment, we needed to connect it to some particular system. We chose to integrate the Interaction Analyzer with Panoply. Panoply is a middleware framework to support ubiquitous computing applications. While the work described here treats the Interaction Analyzer in the Panoply context, we should emphasize that, with relatively little effort, the Analyzer could be integrated with other types of ubiquitous computing system. The suitability and ease of the port will depend on the degree to which the target system relies on a message-based paradigm for interactions, since that is what the Analyzer itself expects.

While the key elements of the Interaction Analyzer do not depend on Panoply constructs, understanding how we used it in the Panoply context requires a little knowledge of how Panoply works. The core representational unit of Panoply is the *Sphere of Influence*, which can represent an individual device or a group of devices united by a common interest or attribute such as physical location, application, or social relationship. Spheres unify disparate notions of "groups," such as device clusters and social networks, by providing a common interface and a standard set of discovery and management primitives.

Panoply provides group management primitives that allow the creation and maintenance of spheres of influence, including discovery, joining, and cluster management. A publish/subscribe event model is used for intra- and inter-sphere communication. Events are propagated between devices and applications, subject to scoping constraints embedded in events of interest. Every sphere scopes policy

and contains a policy manager [5] that monitors the environment, mediates interactions and negotiates agreements.

Panoply supports the design of applications that express their needs and communicate through events. Panoply applications (e.g., the Smart Party) can create custom events, and designate the scope and destination of such events. More details on Panoply can be found in [6].

For the purpose of understanding the Interaction Analyzer, one can regard Panoply as a support system for applications made up of discrete, but interacting, components at various physical locations. These components communicate by message, and generally run code in response to the arrival of a message. Code can also be running continuously or periodically, or can be triggered by other events, such as a sensor observing a real-world event.

Several applications have been built for Panoply [5], [6], [7], and the Interaction Analyzer has been used to investigate many of them. We will concentrate our discussion of the Interaction Analyzer's use on one Panoply application, the Smart Party [7], touching more lightly on its use for other applications.

In the Smart Party, a group of people attends a gathering hosted at someone's home. Each person carries a small mobile device that stores its owner's music preferences and song collection. The party environment consists of a series of rooms, each equipped with speakers. The home is covered by one or more wireless access points. **Figure 1** shows the configured version of a Panoply Smart Party, in which three rooms in a house are capable of playing music and party attendees with various different musical preferences are located in each room.

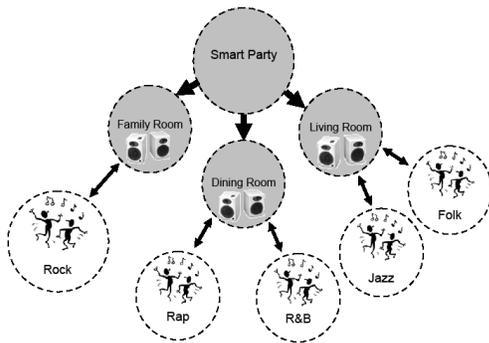


Figure 1. A Panoply Smart Party

As each guest arrives, his mobile device automatically associates with the correct network to connect it to the Smart Party infrastructure. As party attendees move within the party environment, each room programs an audio playlist based on the communal music preferences of the current room occupants and the content they have brought to the party. For example, for the party in **Figure 1**, rock music would play in the family room, since the guests there all have that preference, while folk or jazz would play in the living room.

A Smart Party room determines which guests are present because they have enrolled automatically in a Panoply sphere belonging to that room, triggered by wireless network enrollment. The Panoply sphere controlling the Smart Party in that room periodically queries the devices of the users in that room for their music preferences. These preferences are currently expressed as particular songs the user would like to hear played. The Panoply sphere then uses the combined responses and a voting procedure [8] to select a song from among those suggested by the users' devices. That song is downloaded to the room (from the user's device or somewhere he specifies) and played, after which the process repeats.

As guests move from room to room, the underlying Panoply framework notices their movements and removes them from their old room, adding them to the new one. Thus, each room's playlist adjusts to the current occupants and their preferences.

The Smart Party is a real, working application, extensively tested in our labs.

The Smart Party application could fail in many ways. It could overlook users, or it could localize them into the wrong rooms. It could fail to obtain preferences from some users. Its algorithms for song selection could be flawed, resulting in endless repetitions of the same song. It could unfairly disadvantage some users in the selection. These are just a few of the many possible causes of failures. Because it must take into account user mobility, and even the possibility of users leaving the Smart Party in the middle of any operation, flawed code to handle dynamics can lead to multiple problems. These characteristics, which caused a good deal of difficulty in getting the Smart Party to operate properly, are likely to be common to a wide range of ubiquitous computing applications. Therefore, the Smart Party is a good representative example of the complexities of debugging such applications.

The problems we actually encountered during the development of the Smart Party application included music playing in rooms with no occupants, failure of some Smart Party components to join the application, and race conditions that sometimes caused no music to play when it should. These and other bugs in the Smart Party were attacked with the Interaction Analyzer. The results will be presented in Section IV.

III. THE INTERACTION ANALYZER

A. Basic Design Assumption

The Interaction Analyzer was designed to help developers debug their applications. Therefore, it was built with certain assumptions:

- The source code for the application is available and can be altered to provide useful information that the Interaction Analyzer requires.
- The system was intended for use during application development, not ordinary application use. This assumption allowed us to rely on the presence of more capable devices (with greater

storage capacity and processing power, for example) than might be available in real deployment.

- Knowledgeable developers would be available to use the recommendations of the Interaction Analyzer to find bugs. The Interaction Analyzer does not pinpoint the exact semantic cause of a bug, but guides developers in quickly finding the element of the system, hardware or software, that is the root cause of the observed problem. Also, this assumption meant that we did not need to provide descriptions of problems that would be meaningful to naïve users unfamiliar with Panoply or the design of the application.

The Interaction Analyzer works on applications that have been specially instrumented to gather information that will prove useful in the debugging process. This instrumented application is run in a testing environment, gathering data as the application runs. The data is stored and organized automatically for use during debugging, if necessary. When developers observe a bug that they need to diagnose, they stop the application and invoke the Interaction Analyzer on the information that has been saved during the run.

The Interaction Analyzer is not intended to find bugs on its own. Rather, it assists developers in finding and understanding the causes of observable bad or unexpected application behaviors. The Interaction Analyzer is not intended as a replacement for tools that perform automated analysis of source code, but as a tool for diagnosing problems with application behavior.

The instrumented code is wrapped by a conditional statement that checks the value of a predefined boolean constant. By altering this value, the instrumented code can be easily removed in the final release of the binary.

The Interaction Analyzer was designed for use in a Linux environment, and is implemented in C. It was designed for debugging programs written in C or C++. It could be ported to other environments with reasonable ease.

B. Protocol Definitions and Execution Histories

The Interaction Analyzer uses a *protocol definition* (which specifies how the application is expected to work) and an *execution history* (which describes what actually happened in the run of the application) to debug applications. Each of these is a directed graph of *events*, where an event corresponds to some interesting activity in the execution of the system. Developers instrument their code to indicate when events occur and to store important information about those events. An event can be primitive or high-level. High-level events are typically composed of one or more primitive events, as specified by the developer.

The Interaction Analyzer uses both temporal order (one event occurring before another) and causal order (such as the event that sends a message must precede the event that receives the message) of events to build the execution history of an application's run. Some of these relationships are found automatically by the Interaction Analyzer's

examination of the source code, while others must be provided explicitly by the developers using instrumentation tools. By recording all events that occur during the execution of a system and their causal relationships, one can reconstruct the image and the detailed behavior of the running system at any time [9].

The protocol definition describes how the system should react and behave in different situations. We store the protocol definition in an event causality graph format. The protocol definition is produced at design time, and the execution history is produced at run time.

C. Creating the Protocol Definition

The protocol definition is a model of the application's expected behavior. Such modeling is always an essential part of a large software project, and is helpful in smaller projects, as well. Models help software developers ensure that the program design supports many desirable characteristics, including scalability and robustness [10]. The Interaction Analyzer requires developers to perform such modeling using UML, a popular language for program modeling. We added some additional elements to the standard UML to support the Interaction Analyzer's needs, such as definitions of protocol events and relation definitions. We modified a popular graphical UML tool, ArgoUML [11], to create a tool called Argo-Analyzer that helps developers build their protocol definition.

The Argo-Analyzer is itself a complex system. Briefly, developers use this tool to specify an application's objects, the relationships between them, the context, and the kinds of events that can occur in a run of the application.

The application is organized into objects. Object types are defined using the Argo-Analyzer. For source code written in OOP languages (such as Java), the classes correspond to the object types. These object definitions are used to organize the protocol definition and describe interactions between different application elements.

Relationship definitions describe relationships between objects. The Argo-Analyzer supports commonly used relationships such as parent-child, as well as other user-defined relationships.

Event templates define the properties of an instance of an important event in the application. There must be an event template for each type of event in the application. The Interaction Analyzer will use these templates to match an execution event with an event in the protocol definition. For an event to match a template, not only must their event type and parameter fields match, but their causality requirements must also match. If the execution event does not have the same kinds of preceding events as the template, it will not match.

The developer uses these and a few other UML-based elements to specify the protocol definition, which describes how he expects his application to work. This definition is, in essence, a directed graph describing causal chains of events that are expected to occur in the application.

Serious effort is required to create the protocol definition, but it is a part of the overall modeling effort that well-designed programs should go through. As with any

modeling effort, the model might not match the actual instantiation of the application. In such cases, an execution history will not match the protocol definition, requiring the developer to correct one or the other. In practice, we found that it was not difficult to build protocol definitions for applications like the Smart Party, and did not run into serious problems with incorrect protocol definitions. Mismatches between definitions and executions were generally signs of implementation bugs, which generally should be fixed even if they do not instantly cause incorrect behavior.

D. Creating the Execution History

There is one protocol definition for any application, but each execution of that application creates its own execution history. The Interaction Analyzer helps direct users to bugs in particular runs of the application by comparing the execution history for that run to the expected execution described in the protocol definition.

Each event in the application should generate a record in the execution history. There are three ways to collect the system information required to create such records that describe an execution history of a program: software, hardware and hybrid. For the Interaction Analyzer, software monitoring was used since it provides more flexibility and does not need extra hardware support.

The monitoring could have been based on external observation of the application's behavior, which would have had the advantage of not requiring any instrumentation of the application. We would have needed some way to observe the scheduling of events, which would have involved observing messages being sent between objects in the system. Inter-machine messages could have been sniffed off the wire (though use of cryptography would have complicated this approach). Messages that did not cross real network boundaries would have been more challenging to capture. In either case, obtaining information about the state of the sending and receiving objects would have been difficult.

We chose instead to gather information about the execution history by instrumenting the application. This approach provides greater detail and produces more powerful execution traces than external monitoring could provide. It does so at the cost of changing the application source code. However, since the target use of the Interaction Analyzer is by application developers during their development and debugging process, the costs were of less concern, and the benefits more compelling.

We provide a library to help with this instrumentation process. This general-purpose Java library provides an interface to generate different kinds of event records and their important attributes and parameters. An application generates an entry in its execution history by calling a method in this library. Doing so logs the entry into a trace file on the local machine. Applications can also define their own kinds of events, which the library can also log. Panoply itself logs its own special kind of events, such as "sphere joins," using this mechanism.

A typical analyzer record contains several fields, including a unique ID for the event being recorded, a developer-defined ID, information on the producer and consumer of the event (such as the sender and receiver of a message for a message-send event), timestamps, pointers to all events that directly caused this record's event, and various parameters specific to the particular kind of event being recorded. Most of the parameters are defined by the application developers, who can also add more parameters if the standard set does not meet their needs.

Adding the code required to record an analyzer event costs about the same amount of effort as adding a printf statement to a C program. For example, the command to generate an event in the Smart Party application under Panoply looks similar to this:

```
PanoplyLogger.logPanoplyObjectCreated(codeID
,panoply-specificEvent,creator,createdObject,
directCauses,additionalParameters);
```

Compare this to an actual print statement that the Smart Party developers used before the Interaction Analyzer was available:

```
Debug.println(ModuleName,Debug.DETAIL,
panoply-specificEvent.EventType + " "+panoply-
specificEvent.EventSubType+"
+additionalMessage);
```

The two statements are of similar length and complexity, and require that the developers provide roughly the same information. However, the old version merely allowed a message to be printed, while the Interaction Analyzer version allowed much more, as will be discussed later.

Typically, all statements that record information on the execution history for the Interaction Analyzer are bracketed by compiler commands to include or exclude them, depending on a compile-time option. Thus, a final recompilation when debugging is finished produces a version of the code without any overheads related to the recording of event history.

Panoply applications run on virtual machines, one or more on each participating physical machine. Each virtual machine can run multiple threads, and each thread can generate and log execution events to a local repository using the Event Analyzer's Execution History Generator component. When a run is halted, the Log Provider component on each participating physical machine gathers its portion of the execution history from its local virtual machines and sends this history to a single Log Collector process running on a centralized machine. When all logs from all machines have been collected, the Log Collector collates them into a single merged execution history.

E. Using the Interaction Analyzer

After developers have created the protocol definition, instrumented their code to build the execution history, and run the instrumented application, they are likely to observe bugs or unexpected behaviors during testing. This is when the Interaction Analyzer becomes useful. Upon observing behavior of this kind, the developer can halt the

application, gather the execution logs (with the help of the Log Collector), and feed them into the Interaction Analyzer. The Interaction Analyzer makes use of both the protocol definition and the execution history.

The Interaction Analyzer is a graphical tool that was built using an internal frame model where the main window contains multiple sub-frame-like windows of two types:

1. System-type windows: These windows are created by default and support the major functions of the Interaction Analyzer.
2. User-type windows: These windows are created (and destroyed) by the developer who is using the Interaction Analyzer for debugging. Typically, each user-type window contains information about particular events or objects in the protocol definition or the execution history.

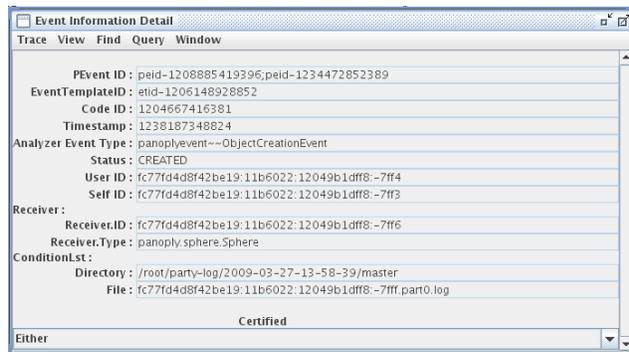


Figure 2. Screenshot of the Interaction Analyzer

Figure 2 is a snapshot of what use of the Interaction Analyzer looks like when the developer starts it. At this point, no execution history has yet been loaded, so all the windows are generic to the application in the abstract, rather than being specific to the erroneous run being debugged. Using a menu option, the user would choose the execution history describing the buggy run he wishes to analyze, at which point the windows would become populated with information specific to that run, and the developer could start to work.

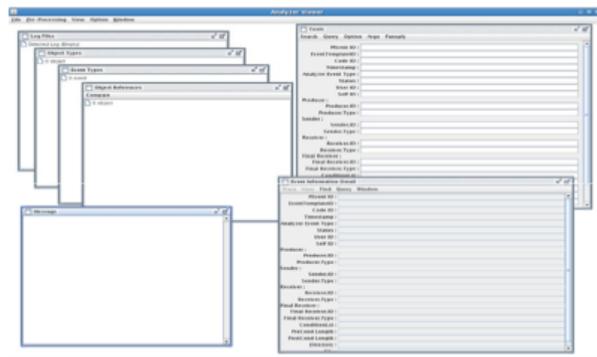


Figure 3. A Sample User-Type Window for an Event

As debugging proceeds, the developer opens and closes windows and navigates between them to assist in tracking down the problem he has observed. Figure 3 is an example of one user-type window that describes an event from the execution history. In this case, it is a Panoply event that has created a sphere. The window shows various event parameter values, such as when the event occurred and what type it was. The Interaction Analyzer will suggest events that are particularly likely to be helpful in debugging various problems, and the user performing the debugging might open this window to help him determine whether there was an obvious error in this particular event.

When a developer opens such an event window, he can take various actions. For example, if the Interaction Analyzer has suggested that this event might be the cause of the error, the developer can investigate the event and, if he determines it is correct, he can validate the event. That action tells the Interaction Analyzer that it should offer a different event as the possible cause. Alternately, the user can ask to see upstream events, perhaps because he suspects that the error that was observed here originated further back in the execution trace, or because he needs more context to understand what should be going on in this event. He can view events at different hierarchical levels, diving down for more detail or popping up to see a higher-level picture of the sequence of events. Similarly, he can ask for downstream events to see what this event led to.

Another option is to find the matching event description in the protocol definition. This option would allow the developer to compare what the protocol said should happen to what actually occurred for this event. Protocol events are described by a similar window, and allow similar kinds of actions: navigation forward and backward, changing of hierarchical levels, obtaining more detail, and so on.

The Interaction Analyzer allows the developers to obtain answers to a number of useful debugging questions, including:

1. Why did an event E not occur?
2. Why did an incorrect event E occur?
3. What are the differences in behavior between objects of the same type?
4. Why did an interaction take a long time?

The developer asks these questions from one of the system-type windows created when the Interaction Analyzer starts execution. For example, to ask a question of Type 1, the developer would specify the event ID of the protocol event he expected to see, but did not, in a field in the Tools system-type window, which is the window in the upper right of Figure 2.

Each of the types of questions that a developer can ask requires somewhat different support from the Interaction Analyzer. We will concentrate on how it addresses questions of Type 1 and 2. The Interaction Analyzer also supports searching for particular execution events and protocol events.

1) Type 1 Questions

Type 1 questions are about why something did not happen when it should have. For example, in the Smart Party, if a user is standing in one of the rooms of the party

and no music is playing there at all, developers want to know “why is no music playing in that room?” There are several possible reasons for this bug. Perhaps the user is not recognized as being in that room. Perhaps the user’s device failed to receive a request to provide his music preferences. Perhaps the room was unable to download a copy of the chosen song from wherever it was stored.

The Interaction Analyzer handles Type 1 questions by comparing the protocol definition and the execution history to generate possible explanations for the missing event. The protocol definition describes event sequences that could cause an instance of that event. The execution history shows the set of events that actually happened, and usually contains partial sequences of events matching the sequences derived from the protocol history. The Interaction Analyzer determines which missing event or events could have led to the execution of the event that should have happened. These sequences are presented to the developer, ordered by a heuristic. The heuristic currently used for presenting possible descriptions of missing events is, following Occam’s Razor, to suggest the shortest sequence of missing events first. The developer examines the proposed sequence to determine if it explains the missing event. If not, the Interaction Analyzer suggests the next shortest sequence.

As a simplified example, say that music is not playing in a room in the Smart Party when guests are present there. The missing event is thus “play music in this room.” The developer performing the debugging will ask a Type 1 question focused on why the “play music” event did not occur in this instance. Complicating factors include the fact that, in the same run, music might have been properly played in other rooms, or even previously or subsequently in the room in question. Thus, the Interaction Analyzer offers methods of specifying the particular context in which debugging should proceed. In this case, the context is the room where the music didn’t play, at the moment when silence was noticed.

The Interaction Analyzer will compare the sequence of events in the actual execution where music did not play to the protocol definition. It might come up with several hypotheses for why music did not play. For example, perhaps the guest who selected a song failed to send it to the player. Or the module that gathers suggestions might have failed to ask any present guests for recommendations. Or the guests might not have been properly recognized as being in that room at all.

There are many possible approaches to determining the relevance of different possible explanations, which then guides where the Interaction Analyzer directs the developer. As mentioned, the Interaction Analyzer currently chooses the explanation with the shortest path, where the path length is defined as the number of events to be added or removed to resolve the problem. In this example, the first of these three explanations (that the guest failed to send the song to the player) requires the fewest “missing events” to serve as an explanation, so it would be investigated first.

The actual methods used by the Interaction Analyzer are more complex [12], since links in the protocol definition and execution history can have AND and OR relationships. Further, real protocols tend to be multilayered and complex. In the case under discussion, for example, sub-protocols are used for user localization, voting, and file transfer. The error could have arisen in any one of these lower-level protocols, in which case eventually the developer would need to move down from the high-level protocol that deals with Smart Party concepts, like asking users for music preferences, to the low-level protocol that might control the transfer of a large file from one or several places to a destination. The Interaction Analyzer understands the concept of multi-layer protocols and offers tools for navigating up and down through these layers.

Further, the Interaction Analyzer makes use of contextual information defined in the protocol definition and recorded in the execution history. For example, if a Smart Party supports music played in several different rooms, a question about why music did not play in the living room will not be matched by events that occurred in the kitchen. The developer performing the debugging will need to specify the context he cares about, since the Interaction Analyzer itself does not know that an event that should have occurred in the living room did not, and thus cannot specify that the location context is the living room. As the developer navigates through the execution graph using the Interaction Analyzer’s advice, he is able to refine his search with further contextual information.

2) Type 2 Questions

Type 2 questions are about why an incorrect event occurred. In the Smart Party context, such questions might be “why was Bill localized in the dining room instead of the family room?” or “why did music play in the entry hall when no one was there?” Type 2 questions are thus about events that appear in the execution history, but that the developer feels do not belong in the history, or have some incorrect elements about their execution.

The Interaction Analyzer works on the assumption that errors do not arise from nowhere. At some point, an event in the application went awry, due to hardware or software failures. The Interaction Analyzer further assumes that incorrectness spreads along causal chains, so the events caused by an incorrect event are likely to be incorrect themselves. If a developer determines that some event is incorrect, either that event itself created the error or one of the events causing it was also erroneous. Working back, a primal incorrect event caused a chain of incorrect events that ultimately caused the observed incorrect event. The developer must find that primal cause and fix the bug there.

Given these assumptions, the job of the Interaction Analyzer in assisting with Type 2 questions is to guide the developer to the primal source of error as quickly as possible. A standard way in which people debug problems in code is to work backwards from the place where the error is observed, event by event, routine by routine, eventually line by line, until the primal error is found. However, this approach often requires the developer to

check the correctness of many events. In situations where the execution of the program is distributed and complex (as it frequently is for ubiquitous applications), this technique may require the developer to analyze a very large number of events before he finds the actual cause of the error.

Is there a better alternative? If one has the resources that the Interaction Analyzer has, there is. The Interaction Analyzer has a complete trace of all events that occurred in the application, augmented by various parameter and contextual information. Thus, the Interaction Analyzer can quickly prune the execution history graph of all events that did not cause the observed erroneous event, directly or indirectly, leaving it with a graph of every event in the execution history that could possibly have contained the primal error. The question for the Interaction Analyzer is now: in what order should these events be analyzed so that the developer can most efficiently find that primal error? Eventually, the developer will need to perform some amount of detailed analysis of code and execution data, but can the Interaction Analyzer help to minimize how much of that analysis is required?

In the absence of information about which events are more likely than others to have run erroneously (which is generally the case), any event in this pruned graph is equally likely to be the source of the error. Assume this graph contains N events. The final event where the error was observed is not necessarily any more likely to be the true source of the error than any other. After all, one of its predecessor events could easily have run erroneously, with the error propagating and only being discovered at this point. If the developer examines the observed incorrect event first, and it was not the source of the error, only one of N possibly erroneous events has been eliminated from the graph.

What if, instead, the Interaction Analyzer directs the developer to analyze some other event E chosen from the middle of the graph? If that event proves correct, then all events that caused it can be eliminated as the source of the primal error. Event E was correct, so the observed error could not have “flowed through” event E ; thus the source of our error is not upstream of E . It must be either downstream or in some entirely different branch of the graph. If event E is erroneous, and E is one of the initial events of the application (one with no predecessor events in the graph), then E is identified as the root cause. If E is not one of the initial nodes, then it is on the path that led to the error, but is not necessarily the original cause of the error. We can then repeat the algorithm, but with event E as the root of the graph, not the event that the developer originally observed, and we continue this process until we find the root cause.

With a little thought, one realizes that the ideal choice of the first event to suggest to the developer would be an event which, if it proves correct, eliminates half of the remaining graph from consideration. If such an event proves incorrect, it eliminates the other half of the graph, since either this event or something upstream must be the root cause, not anything downstream. (There is an assumption here that errors do not simply “go away.”

Thus, if we are examining event E because an erroneous event downstream of E was observed, and the event E is also erroneous, the Interaction Analyzer assumes that the path of error flowed through event E .) If no event whose examination can eliminate half the graph can be found, due to the shape of the graph, then selecting the event whose analysis will eliminate as close to half of the graph as possible is the right choice.

The Interaction Analyzer uses this heuristic to select events for developer investigation. After pruning irrelevant events from the execution history graph, it directs the developer to investigate the event node in that graph whose elimination would most nearly divide the remaining graph in half. The algorithm proceeds as suggested above, eliminating roughly half of the remaining nodes at each step, and eventually the highest erroneous event in the graph is identified as the root cause.

The algorithm stops when it finds this event. It assumes that the incorrectness of the event causes the observed problem that led to debugging, and thus it is the root cause. That is not necessarily true. Consider event X caused by events Y and Z . We observe some incorrect behavior in X , examine Y , and determine it is incorrect. The algorithm would report Y as the root cause, if nothing upstream of Y is incorrect. However, it’s possible that the real problem is in event Z , which the algorithm never suggested examining.

However, if the incorrectness of the event reported did not actually cause the observed problem, then it is the root cause of a different problem. In other words, this algorithm will find the root cause of either the original problem detected by the developer or another problem that he does not know of a priori. If the developer can determine that this event, despite being incorrect, could not have caused the observed behavior, he can run the Analyzer again, this time indicating that this incorrect event actually is correct. This will cause the Analyzer to look elsewhere, and thus to find a different root cause of the original problem. In the example above, having determined that, despite its incorrectness, event Y could not have caused the observed misbehavior in event X , a second run of the Interaction Analyzer with Y marked as correct would lead us to event Z , the true cause of the problem.

At each step, the developer manually investigates one event and tells the Interaction Analyzer whether that event is correct. But by using this technique, the developer need not work his way entirely up the whole execution history graph until he finds the problem. In general, the Interaction Analyzer allows the developer to perform the debugging with few human analysis steps. (In four real cases, using the Analyzer required examination of 4-12 events, out of 200-21,000 total events, depending on the case. Some detailed examples will be discussed in Section IV B.) As long as the Interaction Analyzer’s automated activities (building the graph, analyzing it to find the next event to recommend, etc.) are significantly cheaper than a human analysis step, this process is much faster and less expensive than a more conventional debugging approach.

IV. EVALUATING THE INTERACTION ANALYZER

There are several aspects to the performance of the Interaction Analyzer that should be addressed in determining its value. We outline them here.

The Interaction Analyzer will be of most value in diagnosing problems in large complex ubiquitous environments where many events and possible causes of problems muddy the waters. In such situations, though, there are obvious questions about whether the Interaction Analyzer can perform sufficiently well at that scale. It will need to gather and analyze a great deal of information, and perhaps the costs of doing so will be too high for practical use. Therefore, we performed various simulation experiments to investigate the Interaction Analyzer's performance on large execution graphs.

Ultimately, the Interaction Analyzer is valuable if it can actually help ubiquitous computing application developers find tricky problems in their systems. To demonstrate the Interaction Analyzer's promise in meeting this fundamental goal, we present case studies involving the actual use of the Interaction Analyzer in finding bugs in the Smart Party application and a second smaller application.

A secondary, but important, practicality aspect of the Interaction Analyzer is its basic performance overheads, so we present data on those overheads, as well.

A. Simulation Results

To determine how the Interaction Analyzer would perform when handling large execution graphs, we generated artificial execution graphs of varying sizes and properties (such as the branching factors in the graph). Erroneous events and their root causes were generated randomly. The results are too extensive to report completely here (see [12] for full results), but some example graphs will show the actual benefits of using this tool and the value of the algorithms it uses to find events for developers to examine.

For each point in the simulation figures shown here, 20 different rooted execution graphs were generated randomly. For each generated graph, 10 different scenarios were generated randomly for different root cause nodes. For a given execution graph and root cause, most of the tested algorithms are deterministic, except for the Terminal Walk algorithm (described below). Even for that algorithm, only one simulation was performed for a given graph and root cause. Thus, each point in the figures represents 200 different runs. The value at each point in the figures is the mean value of the number of validation requests for all scenarios. The size of the graph in all simulations represents the rooted graph size. In other words, we assume all irrelevant events that could not directly or indirectly cause the detected incorrect event are already pruned from the graph. The confidence level is 95%. In some cases, the variation is small enough that the error bars are essentially invisible.

A major question for the Interaction Analyzer is how it chooses which event to suggest for further investigation.

When looking for a Type 2 error ("why did this incorrect event occur?"), one could examine the graph of all events that directly or indirectly caused the erroneous event and randomly choose one to recommend for examination. Unless some nodes are more likely to be erroneous than others, randomly selecting one of the nodes to examine is just as likely to pinpoint the root case as walking back step-by-step from the observed error, which is a traditional debugging approach. For reasons not important to this discussion, we have termed the algorithm that randomly selects a node from the graph "Terminal-Walk." Results for the Terminal-Walk algorithm thus also describe the traditional approach, under the assumption that any event in the pruned graph is equally likely to be the original source of error.

The algorithm that the Interaction Analyzer actually uses (see Section III.E.2) analyzes the portion of the execution graph associated with the erroneous event and directs the developer to an event whose correctness status will essentially eliminate half the nodes in this graph, as described earlier. We term this approach the "Half-Walk" algorithm.

Figure 4 shows the performance of these algorithms for event graphs of different sizes. The x-axis parameter refers to the number of nodes in the pruned causal graph rooted at the observed erroneous event, any one of which could be the root cause of the observed error. The x-axis is a log scale. The "Validation Cost" is in number of events, on average, that the developer will need to examine by hand to find the error. The first graphs we discuss have uniform branching factors (i.e., the branching factor of each node in the graph is randomly chosen from a uniform distribution, with each branch indicating an event caused by the event described by this node). The probability that each node in the graph (including the node where the error was observed) is the root cause of the error is generated from a uniform distribution. The actual root cause is then randomly selected following that probability distribution. The Terminal-Walk algorithm is unaware of this probability distribution, and merely randomly selects one of the possible events from the graph for examination.

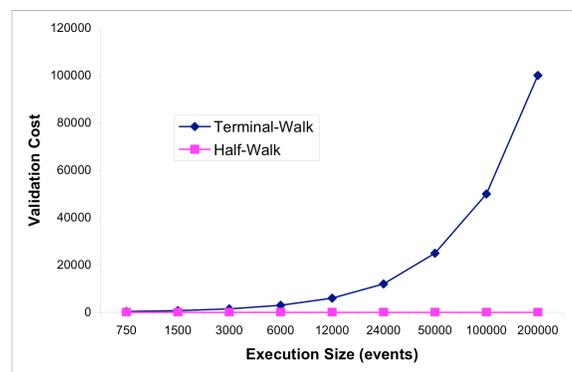


Figure 4. Terminal-Walk vs. Half-Walk Algorithm

The Terminal-Walk algorithm becomes expensive as the number of potential causes of the observed error grows. Each validation represents a human developer examining code and state information for an event in the system, which is likely to take at least a few minutes. The Half-Walk algorithm, on the other hand, is well behaved, displaying \log_2 behavior.

In some situations, the probability of failure in each event is known. For example, the system may consist of sensors with a known rate of reporting false information. Even if event failure probabilities are not perfectly known, an experienced developer may have a sense of which events are likeliest to be the root cause of errors.

If the developer has perfect knowledge of the probability that each event was performed correctly, he might use an algorithm that first examines the event with the highest probability of being correct. If that event is indeed correct, he could eliminate from further consideration all events that caused that event. He could then move down the list of probabilities as candidates are eliminated. We term this algorithm the Highest-Walk algorithm.

Figure 5 shows the relative performance for the Highest-Walk algorithm vs. the Half-Walk algorithm (which the Interaction Analyzer actually uses) for graphs and root causes of the same kind shown in Figure 4. If we have this knowledge of probability of event failure, the Half-Walk algorithm is altered so that, instead of selecting an event that eliminates half the nodes in the graph, it selects an event that eliminates half the probability of error in the graph. Highest-Walk is, unlike Terminal-Walk, competitive with Half-Walk, but Half-Walk is clearly better. For 200,000 events in an execution graph, Half-Walk will require the developer to examine less than half as many events as Highest-Walk would. The probability of being incorrect is propagated down the event path, and thus the event with highest probability of being incorrect is normally very far from the root cause. Thus, the Highest-Walk algorithm does not perform as well as Half-Walk.

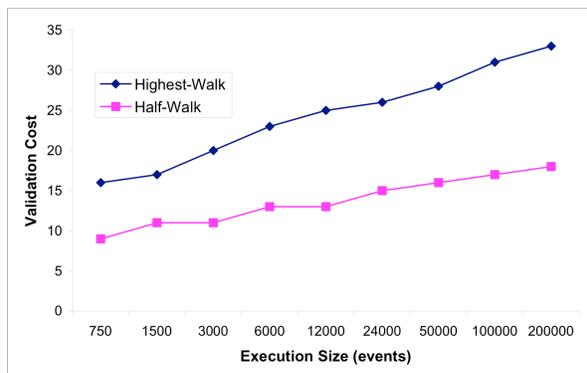


Figure 5. Highest-Walk Algorithm vs. Half-Walk Algorithm

Many factors can cause variation in the performances of these algorithms. For example, the distribution of branching factors can be varied, where branching factor describes how many events are caused by a given event. The figures already discussed assumed a branching factor for each node chosen from a random distribution. Perhaps, instead, there is a distribution of branching factors randomly generated in a linear scale, where nodes with a lower branching degree appear more often in the event graph. Thus, branching factors of 1 are more probable than branching factors of 2, which are more probable than branching factors of 3, and so on.

Figure 6 illustrates the relative performance of the Highest-Walk and Half-Walk algorithms in this case. Since the distribution used here favors small branching factors, generally a node will have a lower branching factor in these graphs than in those discussed earlier. Here, the Highest-Walk performs much worse than the Half-Walk because the Highest-Walk chooses the node with highest probability of being correct for validation. Consequently, after each validation, the new size of the event graph (in the Highest-Walk) tends to be large. In other words, it prunes the tree less effectively. With uniform branching (as in Figure 5), the graph is more balanced; whereas in linear branching, the graph is less balanced. With uniform branching, we are less likely to find nodes with a really high probability of being incorrect. Thus, in uniform branching, the Highest-Walk has better performance, though it is still inferior to the Half-Walk algorithm.

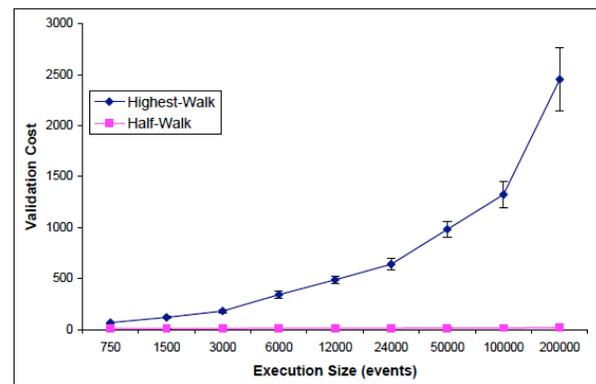


Figure 6. Constant Root Cause Factor – Linear Branch

B. Case Studies Using the Interaction Analyzer

Simulation studies are helpful in understanding the Interaction Analyzer's behavior in many different circumstances, but ultimately the point of a debugging tool is that it prove helpful in solving real problems. In this section we describe how the Interaction Analyzer helped us find real bugs in real applications. The primary application discussed is the Smart Party application introduced in Section II. This application was not written to help us investigate the behavior of the Interaction Analyzer. On

the contrary, the Interaction Analyzer was built to help us debug problems with the Smart Party and other Panoply applications.

We also present a case study for a second real application, one that is much smaller. Our graduate students grew tired of having to get up to open the locked door to our lab when someone without a key knocked on it, so they designed a bit of hardware and a small Panoply application that would automatically unlock the door under certain conditions. For example, if a knock was heard during regular working hours and someone was actually in the lab, the door should open. Also, users in the room with sufficient privilege could simply order the door to open. This application was vastly simpler than the Smart Party, but it was also real working Panoply software, and exhibited real bugs.

1) *Smart Party Bug 1: Music Playing in the Wrong Room*

This bug occurred in the Smart Party when the application was run with three rooms and one user. Music played in a room where no user was present. Before availability of the Interaction Analyzer, the developers of the Smart Party had used traditional methods to find the root cause of this problem, which proved to be that the user location determination module had put him in the wrong room. We did not keep records of how long the debugging process took before the Interaction Analyzer was available, but it was far from instant.

This was a Type 2 error, an event occurring incorrectly. As mentioned in Section III, the Interaction Analyzer uses contextual information when available to guide the process of finding root causes. We investigated this bug both with and without contextual information. Without contextual information, the Analyzer had to suggest six events (out of a possible 8000 in the execution history) to pinpoint the problem. With contextual information (the developer indicating which room he was concerned about), the Interaction Analyzer found the problem in one step.

2) *Smart Party Bug 2: No Music Playing*

This bug occurred in some, but not all, runs of the Smart Party. A user would join the Smart Party, but no music would play anywhere. Since this bug was non-deterministic, it was extremely difficult to find using standard methods. In fact, the Smart Party developers were unable to find the bug that way.

Once the Interaction Analyzer was available, it found the bug the first time it occurred. This was a Type 1 error, an event that did not occur when it should have. The Interaction Analyzer found the root cause by comparing the protocol definition to the execution history and noting a discrepancy. The Interaction Analyzer made use here of its ability to deal with events at multiple hierarchical levels. At the high level, it noted that music did not play and that the high-level protocol definition said it should.

The Analyzer determined that the failure was due to not responding to a request by the user for a localization map. To further determine why that request wasn't honored, the Analyzer suggested to the developer that he dive down to a lower protocol level, and eventually, to an even lower level. The bug ultimately proved to be in the code related to how Panoply routed messages.

The Interaction Analyzer found this bug in three queries, a process that took less than five minutes, including the time required by the developers to examine the code the Analyzer recommended. Without the Analyzer, the developers had been unable to find this bug over the course of several weeks.

3) *Door Opener Bug*

The door opener application described previously had a bug that caused the door not to open when it was ordered to do so. Since the bug appeared to be failure of an expected event, a query of Type 1 was used, and the Analyzer was able to find two different root causes. The first cause was a serial port configuration problem, and thus the application failed to open the port and could not send commands to the hardware controlling the door lock. The second cause was a mistake in the policy describing which spheres should talk to each other, and thus the command could not reach the controller. The first cause was found immediately since it happened in the highest protocol layer. The second root cause was found after going down five protocol layers. There were about 340 execution events in the log files. The Interaction Analyzer recommended that a total of 6 events be examined to find both problems. This case demonstrates both the value of the Interaction Analyzer even for relatively simple applications, and how the Interaction Analyzer can find multiple bugs that cause a single observed problem.

TABLE 1. INTERACTION ANALYZER COSTS

Operation	Example Cost	Average Cost
Import Exec Hist.	3.5 seconds	.35 msec/event
Preprocessing	.3 seconds	.03 msec/event
Load Prot. Def.	7 seconds	.82 msec/element
Matching	12.2 seconds	1.18 msec/event
Total Time	23.0 seconds	2.48 msec/event

C. *Interaction Analyzer Overheads*

Table 1 shows some of the overheads associated with using the Interaction Analyzer. The Example Cost column shows the actual total elapsed times for handling all events in a sample 11,000 event execution history. The Average Cost column shows the normalized costs averaged over 20 real execution histories. These costs are paid every time a developer runs the Interaction Analyzer, and essentially represent a startup cost. For an 11,000 event run, then, the developer needs to wait a bit less than half a minute before his investigations can start.

The other major overhead is the cost for the Interaction Analyzer to respond to a user query. For queries of Types 1, 3, and 4, this cost is less than a second. For queries of Type 2, it depends on the size of the portion of the execution history that is rooted at the event the developer needs to investigate, not the size of the entire history. Any event that exerted a causal influence on the event under investigation must be considered. **Figure 7** shows the time required to choose an event for the developer to evaluate for causal graphs of different sizes. If there are 100,000 events in the causal graph of the investigated event, it takes around 17 seconds to recommend one to the developer. This graph is log scale on the x-axis, so the time is roughly linear as the number of events grows. The Interaction Analyzer chooses an event for validation such that its examination will eliminate around half of the graph; so if the event in question is not the root cause, the second recommendation will be made on a graph of half the size of the original, and thus half the cost.

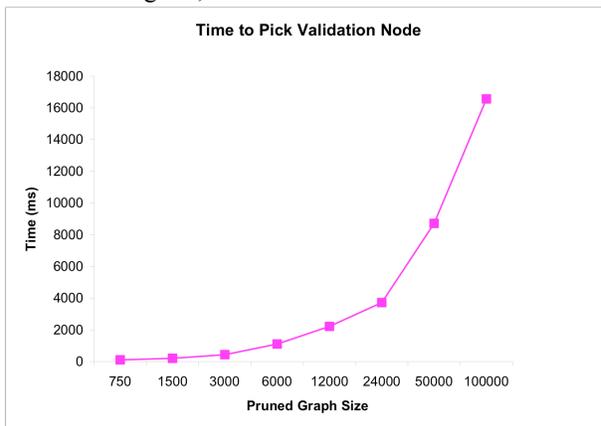


Figure 7. Time to Pick Validation Node

D. Usability and Utility Issues

The goal of the Interaction Analyzer is to ease the burden for developers, so whether it does so is a major concern. We have anecdotal evidence to support its value. In terms of cost, there is a learning curve associated with first using the system and with instrumenting applications. This curve is not steep, in our experience. The actual cost of instrumenting applications is relatively low. The statements that must be inserted into the code are no more complex than more traditional print debugging statements that most programmers already use (see Section III.E).

In terms of run time costs, the Interaction Analyzer is not intended to be run in production mode, so the performance costs are paid only when performing debugging. They are not sufficiently high to slow debugging, since they are no greater than printing log outputs. There is a cost to gather and analyze the data from a run, but this is a matter of a few seconds to a few minutes for a typical run (see Section IV.C).

While these costs are low, they are still not worthwhile unless there is commensurate benefit. Our experiences show that there is, as outlined by the case studies above. The Interaction Analyzer succeeded in finding real bugs that standard debugging techniques could not, even in situations where analysts had spent days or weeks tracking the bugs down. The Interaction Analyzer did so in a matter of a few minutes. It was helpful in finding complex bugs that were non-deterministic and depended on race conditions, situations that are notoriously hard to deal with using standard debugging techniques.

A fuller statement of utility and usability would require detailed human studies and experiments that have not been performed. However, the initial results are promising and suggest that the system is worth far more than its costs.

V. RELATED WORK

Several systems have supported debugging problems in complex distributed systems. The most closely related are those that build execution graphs based on data gathered during a run. RAPIDE [13] was an early system that used this approach. In RAPIDE, an event can be used to denote any action or behavior of a system. By capturing enough events, the image of the application runtime can be reproduced later. The author also proposed hierarchical viewing for event management. RAPIDE aggregates sets of low-level system events into a higher-level event to give information about the component objects at the application level. Different abstraction hierarchies can be used to display the system in different views. RAPIDE also supports event filtering based on a predefined pattern.

RAPIDE was extended to build an execution architecture that captured causal relationships between runtime components [14]. This system creates an image of the running system that helps the developer visualize all interactions and relationships between components during execution. Based on the visual graph, developers can understand the execution architectures of dynamically changing software systems. If the execution architecture is different from the specification, an exception is raised to report the abnormality. This work supports building models of bad behavior to detect known problems.

The Event Recognizer [15] treats debugging as a process of creating models of expected program behaviors and comparing these to the actual behaviors exhibited by the program. The Event Recognizer matches actual system behavior from event stream instances to user-defined behavior models. Incompletely recognized behaviors indicate that the modeler should more closely examine the class of behaviors that are missing, or explain what is wrong with a particular program execution. The tool helps to detect abnormal behavior (that is not defined in the behavior models) and shows how well the actual behavior fits the user-defined pattern. If a bad behavior happens to fit one of the defined behavior models, the system will not be able to detect the problem.

Poutakidis et al. [16] uses interaction protocol specifications to detect interactions that do not follow the

protocol. Interaction protocols are specified using AUML and translated to Petri nets. The debugger uses Petri nets to monitor conversations and detect any unexpected or missing message when interaction does not follow the protocols. More specifically, it can detect failures such as un-initialized agents, sending messages to the wrong recipient, sending the wrong message and sending the same message multiple times. However, it does not explain why the problem occurs and the root cause.

Other approaches use non-graph-based methods to find root causes. Yemini and Kliger [17] treat a set of bad events as a code defining the problem, and use decoding methods to match it to known problems. This approach assumes that the developers know which sets of bad events occur when a problem happens.

Piao [18] uses Bayesian network techniques to determine root causes of errors in ubiquitous systems. The Bayesian network describes a complex system as a compact model that presents probabilistic dependency relationships between various factors in a domain. System real-time performance data is collected, including the system health states. Certain parameters are selected and ranked in a node list that will be applied in structure learning. Bayesian machine learning is applied for topology structure learning to find a network structure that is the most probable match to the training data. This network structure is used to infer the root cause for real-time data from an erroneous run. This approach requires a large training set to build a complete dependency graph and does not work well for a system with a large number of parameters due to the over-fitting phenomenon in machine learning. Therefore, it is more applicable for systems with small sets of parameters, such as a network system described solely by CPU, throughput, RAM use, and bandwidth.

Ramanathan [19] designed a system to find the root causes of errors in sensor networks. This system collects network-related data such as routing table, neighbor list, up time, bad packets received, etc. Based on the specific relationship between these collected data, the system detects failures and triggers localization. For example, the neighbor list and the up time can be used to detect a failed sensor node. Since the collected metrics are sensor-network specific, this approach can only be directly applied for sensor network environments.

Urteaga's REDFLAG system [20] is a fault detection service for data-driven wireless sensor applications. It consists of a Sensor Reading Validity (SRV) sub-service, which detects erroneous sensor readings, and a Network Status Report (NSR) subservice, whose task is to abate data loss by identifying unresponsive nodes. These two subservices help to identify failed sensors in the network.

Gardner [21] proposed a framework to monitor efficiently all system events and information in an operating system, with the goal of providing a detailed look at operating system kernel events with very low overhead. This collected information can be used to analyze problems in the underlying system and provide necessary information so that adaptive applications can adjust. This

framework instruments the kernel and network library code to generate events; and the developers are expected to examine these recorded events by themselves to identify the problem. Also, this system is not distributed, and thus not easily adaptable for a ubiquitous environment.

Hseush's debugging approach [22] concentrates on data, rather than events, arguing that data is a more meaningful way for program users to approach debugging than control flow. In this approach, the user must be aware of data flow as well as control flow and/or message flow. A debugging language is provided to express breakpoints, single stepping and traces in terms of the data as well as the control. For example, concurrent accesses on a shared memory location can be described using the language; and when the user detects a matched behavior to the described concurrent accesses, the application can be stopped or suspended accordingly.

Other researchers have approached debugging of ubiquitous environments and other distributed systems from entirely different angles, potentially offering complementary ways to help diagnose problems in such environments. [23] describes a suite of tools that help visualize multi-agent applications. Each tool provides a different perspective of the application being visualized. For example, a society tool shows the structural organizational relationships and message interchanges between agents in a society, while a report tool graphs the society-wide decomposition of tasks and the execution states of the various sub-tasks. The complete set of tools provides various perspectives on the condition of the distributed application.

[24] and [25] propose different system views that allow a graphical representation of the selected aspects of the system state and its dynamic behavior. An agent view shows the structural or behavioral agent model. An interaction view shows patterns of interactions such as message-passing activity. A cooperation view shows the potential or current task requests between agents. These views represent the developer's conceptual models, such as the agent, distribution and interaction models proposed by the authors.

Such alternate approaches can conceptually be combined with the debugging services offered by the Interaction Analyzer, giving different perspectives from which to view any particular debugging problem. Which set of views and tools is most helpful for actual debugging of complex ubiquitous computing problems is a question for further research.

VI. CONCLUSIONS

Ubiquitous systems are complex, consisting of many different components. Their dynamic nature makes it hard to develop and debug them. Bugs often become evident long after and far away from their actual cause. The Interaction Analyzer provides quick, precise determination of root causes of bugs in such systems. While developed for Panoply, it can be adapted for many ubiquitous computing environments. The Interaction Analyzer has

been demonstrated to have good performance by simulation, and has been used to find actual bugs in real ubiquitous computing environments, including cases where more traditional debugging methods failed.

The Interaction Analyzer is fundamentally a tool to help developers perfect their ubiquitous computing applications. With significantly more work, it could perhaps be adapted to work in deployment scenarios, helping average users fix the problems they observe. Generally, however, the advice the Interaction Analyzer can provide would not be very helpful to a typical user. Much more effort would be required to assist in mapping from low-level problems to solutions that make sense to a typical user. Further, if the problem is rooted in flawed code, rather than a failed hardware component or a mistake in configuration, it is not likely that the user will be able to solve the problem, even if he can pinpoint it. Nonetheless, debugging for ubiquitous computing users is likely to be a problem of increasing importance for the future, and deserves more study.

ACKNOWLEDGMENT

This work was supported in part by the U.S. National Science Foundation under Grant CNS 0427748.

REFERENCES

- [1] N. Nguyen, L. Kleinrock, and P. Reiher, "The Interaction Analyzer: A Tool for Debugging Ubiquitous Computing Applications," *Ubicomm 2011*, November 2011.
- [2] W. Edwards and R. Grinter, "At Home With Ubiquitous Computing: Seven Challenges," *LNCS*, Vol. 2201/2001, 2001, pp. 256-272.
- [3] J. Bruneau, W. Jouve, and C. Counsel, "DiaSim: A Parameterized Simulator for Pervasive Computing Applications," *Mobiquitous 2009*, pp. 1-3.
- [4] T. Hansen, J. Bardram, and M. Soegaard, "Moving Out of the Lab: Deploying Pervasive Technologies in a Hospital," *Pervasive Computing*, Vol. 45, Issue 3, July-Sept. 2006, pp. 24-31.
- [5] V. Ramakrishna, K. Eustice, and P. Reiher, "Negotiating Agreements Using Policies in Ubiquitous Computing Scenarios," *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA'07)*.
- [6] K. Eustice, *Panopoly: Active Middleware for Managing Ubiquitous Computing Interactions*, Ph.D. dissertation, UCLA Computer Science Department, 2008.
- [7] K. Eustice, V. Ramakrishna, N. Nguyen, and P. Reiher, "The Smart Party: A Personalized Location-Aware Multimedia Experience," *Consumer Communications and Networking Conference*, January 2008, pp. 873-877.
- [8] K. Eustice, A. Jourabchi, J. Stoops, and P. Reiher, "Improving User Satisfaction in a Ubiquitous Computing Application," *SAUCE 2008*, October 2008.
- [9] P. Bates, "Debugging Heterogeneous Distributed Systems Using Event-Based Models of Behavior," *ACM TOCS*, Vol. 13, No. 1, February 1995, pp. 1-31.
- [10] The Object Management Group, <http://www.omg.org>, July 12, 2012.
- [11] ArgoUML, the UML Modeling Tool. <http://argouml.tigris.org>, July 12, 2012.
- [12] N. Nguyen, *Interaction Analyzer: A Framework to Analyze Ubiquitous Systems*, Ph.D. dissertation, UCLA Computer Science Department, 2009.
- [13] D. Luckman and J. Vera, "An Event-Based Architecture Definition Language," *IEEE Transactions on Software Engineering*, Vol. 21, No. 4, April 2005, pp. 717-734.
- [14] J. Vera, L. Perrochon, and D. Luckham, "Event Based Execution Architectures for Dynamic Software Systems," *IFIP Conference on Software Architecture*, 1999, pp. 303-308.
- [15] P. Bates, "Debugging Heterogeneous Distributed Systems Using Event-Based Models of Behaviors," *ACM Transactions on Computer Systems*, Vol. 13, No. 1, February 1995, pp. 1-31.
- [16] D. Poutakidis, L. Padgham, and M. Winikoff, "Debugging Multi-Agent Systems Using Design Artifacts: The Case of Interaction Protocols," *1st International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002, pp. 960-967.
- [17] A. Yemini and S. Kliger, "High Speed and Robust Event Correlation," *IEEE Communications Magazine*, Vol. 34, No. 5, May 1996, pp. 82-90.
- [18] S. Piao, J. Park, and E. Lee, "Root Cause Analysis and Proactive Problem Prediction for Self-Healing," *Int'l Conference on Convergence Information Technology*, 2007, pp. 2085-2090.
- [19] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the Sensor Network Debugger," *Int'l Conference on Embedded Networked Sensor Systems*, 2005, pp. 255-267.
- [20] I. Urteaga, K. Barnhart, and Q. Han, "REDFLAG: A Runtime, Distributed, Flexible, Lightweight, and Generic Fault Detection Service for Data Driven Wireless Sensor Applications," *Percom 2009*, pp. 432-446.
- [21] M. Gardner, W. Feng, M. Broxton, A. Engelhart, and G. Hurwitz, "MAGNET: a Tool for Debugging, Analyzing and Adapting Computing Systems," *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid*, 2003.
- [22] W. Hseush and G. Kaiser, "Data Path Debugging: Data-Oriented Debugging for a Concurrent Programming Language," *Proceedings of the 1988 ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging*, Madison, Wisconsin, United States, May 1988.
- [23] D. Ndumu, H. Nwana, L. Lee, and J. Collis, "Visualizing and Debugging Distributed Multi-agent Systems," *Autonomous Agents*, Seattle WA, USA, 1999.
- [24] M. Liedekerke and N. Avouris, "Debugging Multi-agent Systems," *Information and Software Technology*, 1995.
- [25] M. Morris, "Visualization for Causal Debugging and System Awareness in a Ubiquitous Computing Environment," *Adjunct Proceedings of UbiComp*, 2004.

Restoration of Blurred Images Using Revised Bayesian-Based Iterative Method

Sigeru Omatu, Hideo Araki, and Yuka Nagashima

Osaka Institute of Technology, Kyocera Ltd

*Asahi-ku, Osaka, 535-8585, Osaka, Japan, Hirakata, Osaka, Japan, Fushimi-ku, Kyoto, 612-8501, Japan
omatu@rsh.oit.ac.jp, araki@is.oit.ac.jp, naga@sig.cs.osakafu-u.ac.jp*

Abstract—A restoration method of degraded images based on the Bayesian-based iterative method is proposed. An iterative method is developed by regarding the observed degraded images as probabilities and point spread functions as the conditional probabilities. The restored images are estimated by using Bayesian rule. We have proposed two kinds of the iterative method. One is to use the observed images as the initial values of the estimated images when we estimate the point spread function. The other method is to estimate the initial values of the point spread function by using the logarithmic amplitude spectrum of the blurred image. The simulation results show that the proposed methods are effective to restore the blurred images.

Keywords—point spread function, Bayesian rule

I. INTRODUCTION

To make a clear image from degraded image, many enhancement techniques have been developed until now [1],[2],[3],[4],[5]. There are so many papers published until now about restoration algorithms. Those papers can be classified into two categories. One is a model-based approach [2],[3],[4] and the other is a learning-based approach.

The main technique for the former approach is to develop sharpness filters [2],[3]. More masks to approximate a derivative operation have been developed. By using the filters they could enhance the image. But they also enlarge pixel noise. The learning-based approach is to develop an iterative algorithm such that a criterion function could be minimized.

We adopt the second approach to enhance images by using the Bayesian rule. This method was first proposed by [5]. The Bayesian rule reflects optimal estimation in a sense to minimize the cost function under noisy observation and an iterative algorithm was proposed to find the optimal solution [6] and [7]. The algorithm include two parts, the first one is to estimate a point spread function from the estimated image and the second one is to estimate the original image by using the estimated point spread function. Thus, this algorithm might be optimal when the observed image is similar to the original image, that is, in case of a high S/N ratio. Therefore, the results will depend on the initial guesses of point spread function although the previous reports are assumed to be fixed [5],[6],[7].

In this paper, we propose two methods. One is to use the observation image instead of the estimated image when we estimate the point spread function (Algorithm I). The other method is to use an estimated point spread function as the

initial guess of the point spread function in the Bayesian-based iterative procedure.

First, we will show the principle of the Bayesian-based iterative method proposed by Richardson [5]. Then, we will propose two methods. After that the simulation results are illustrated to show the effectiveness of the proposed methods.

II. PRINCIPLE OF IMAGE RESTORATION

In image enhancement, the ultimate goal of restoration techniques is to improve a given image in some sense. Restoration is a process that attempts to recover an image that has been degraded by using a priori knowledge of the degradation phenomenon. As shown in Fig. 1, the degradation process may be modeled as an operator \mathbf{H} in case of noiseless situation. It operates on an input image $f(x, y)$ to produce a degraded image $g(x, y)$. For the sake of simplicity, we denote $f(x, y)$ by $f(x)$, $g(x, y)$ by $g(x)$, $h(x, y)$ by $h(x)$, etc. In equation form, we have

$$g(x) = \mathbf{H}f(x) = h * f(x) = \sum_{y=-\infty}^{\infty} h(x-y)f(y) \quad (1)$$

where $h(x)$ is an impulse response and $*$ denotes the operation of convolution.

Based on the convolution theorem, the frequency domain representation of Eq. (1) becomes

$$G(j\omega) = H(j\omega)F(j\omega) \quad (2)$$

where $G(j\omega)$, $H(j\omega)$, and $F(j\omega)$ are Fourier transforms of $g(x)$, $h(x-y)$, and $f(y)$, respectively. As shown in Fig. 1, given $G(j\omega)$ and some knowledge about $H(j\omega)$, the objective of restoration techniques is to recover $F(j\omega)$ which means to recover the original image $f(x)$ via the inverse Fourier transform.

III. RICHARDSON'S ITERATIVE METHOD

We will review an iterative method by Richardson [5] in this section. Given the degraded image g , the point spread function h and the original image f are estimated based on Bayes' theorem. It will be effective to estimate the original image f from the observed image g . It was assumed that g , h , and f are discrete and are not necessarily normalized. The numerical values of g , h , and f are considered as measures of the frequency of the occurrence of them at those points. h is usually in normalized form. Energy of f originating at a point

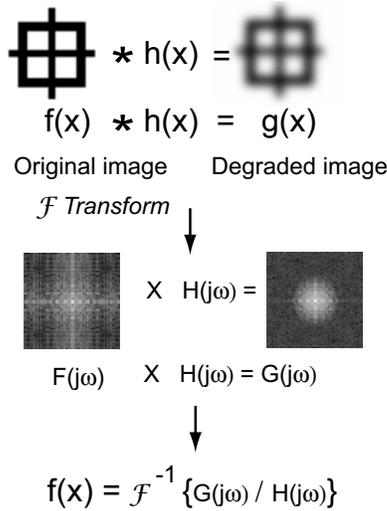


Fig. 1. The restoration principle.

is distributed as g at points according to the energy indicated by h . Thus, g represents the resulting sums of the energy of f originating at all points.

In the notation of this problem the usual form of the Bayes' theorem is stated as the conditional probability of f , given g . It was assumed that the degraded image g was of the form $g = h * f$, where $*$ denotes the operation of convolution such that

$$g(x) = h * f(x) = \sum_{y=-\infty}^{\infty} h(x - y)f(y). \quad (3)$$

Note that f and g are intensity functions of the original image and observed image, respectively and h is the weighting function depending on image measurement devices. We assume that the input image and the weighting function are unknown. The values of f , g , and h are not limited within $[0,1]$. We normalize and denote them by f' , g' , and h' . Thus, we have

$$f'(x) = \frac{f(x)}{\sum_{x=-\infty}^{\infty} f(x)} = \frac{f(x)}{F} \quad (4)$$

$$g'(x) = \frac{g(x)}{\sum_{x=-\infty}^{\infty} g(x)} = \frac{g(x)}{G} \quad (5)$$

$$h'(x) = \frac{h(x)}{\sum_{x=-\infty}^{\infty} h(x)} = \frac{h(x)}{H} \quad (6)$$

where F, G , and H could be equal since the restoration process is conservative. Note that f , g , and h are nonnegative and the total sums are equal to one. Thus, we could regard them as probability measures and $f'(x_1)$ as the probability measure of the original image $f(x_1)$ at x_1 . This means that the possibility of the existing intensity of the original image $f(x_1)$ at x_1 . Similarly, $g'(x_2)$ and $h'(x_1)$ mean the possibility

of the existing intensity of the observed image $g'(x_2)$ at x_2 and the possibility of the transition weight from the input image $f(x_1)$ at x_1 to the output image $g(x_2)$ at x_2 . Therefore, we have

$$P(g(x_2)|f(x_1)) = P(h(x_2 - x_1)) = h'(x_2 - x_1). \quad (7)$$

The above relation can be derived by using the following relation.

$$\begin{aligned} P(g(x_2), f(x_1)) &= P(g(x_2)|f(x_1))P(f(x_1)) \\ &= P(h * f(x_2), f(x_1)) \\ &= P(h(x_2 - x_1), f(x_1)) \\ &= P(h(x_2 - x_1))P(f(x_1)) \end{aligned}$$

where we have used independence assumption between original image and restoration mechanism.

Using the Bayes' theorem we have

$$\begin{aligned} P(f(x)|g(x_2)) &= \frac{P(g(x_2)|f(x))P(f(x))}{\sum_{x_1=-\infty}^{\infty} P(g(x_2)|f(x_1))P(f(x_1))} \\ &= \frac{f'(x)h'(x_2 - x)}{\sum_{x_1=-\infty}^{\infty} f'(x_1)h'(x_2 - x_1)}. \end{aligned} \quad (8)$$

If we multiply the both sides of Eq. (8) by $P(g(x_2)) = g'(x_2)$ and take the summation with respect to x_2 , we get

$$\begin{aligned} P(f(x)) &= f'(x) \\ &= f'(x) \sum_{x_2=-\infty}^{\infty} \frac{h'(x_2 - x)g'(x_2)}{\sum_{x_1=-\infty}^{\infty} f'(x_1)h'(x_2 - x_1)}. \end{aligned} \quad (9)$$

Considering $F = G = H$ and multiplying them both sides of the above equation, we have

$$f(x) = f(x) \sum_{x_2=-\infty}^{\infty} \frac{h(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} f(x_1)h(x_2 - x_1)}. \quad (10)$$

Using the above equation, Richardson[5] proposed the following recurrence procedure to find the original image $f(x)$.

$$\begin{aligned} f_{n+1}(x) &= f_n(x) \sum_{x_2=-\infty}^{\infty} \frac{h_n(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} h_n(x_2 - x_1)f_n(x_1)} \\ n &= 0, 1, 2, \dots \end{aligned} \quad (11)$$

In order to derive the recursive equation of the point spread function $h(x)$, we will set $x_3 = x_2 - x$. Then from Eq. (8) we have

$$P(f(x_2 - x_3)|g(x_2)) = \frac{f'(x_2 - x_3)h'(x_3)}{\sum_{x_1=-\infty}^{\infty} f'(x_1)h'(x_2 - x_1)}. \quad (12)$$

Multiplying both sides of the above equation by $P(g(x_2)) = g'(x_2)$, we have

$$P(f(x_2 - x_3)|g(x_2))P(g(x_2)) = g'(x_2) \frac{f'(x_2 - x_3)h'(x_3)}{\sum_{x_1=-\infty}^{\infty} f'(x_1)h'(x_2 - x_1)}. \tag{13}$$

Using the Bayes' rule, we have

$$\begin{aligned} &P(f(x_2 - x_3)|g(x_2))P(g(x_2)) \\ &= P(f(x_2 - x_3), g(x_2)) \\ &= P(g(x_2)|f(x_2 - x_3))P(f(x_2 - x_3)) \\ &= h'(x_3)f'(x_2 - x_3). \end{aligned} \tag{14}$$

From Eqs. (14) and (13), we have

$$h'(x_3)f'(x_2 - x_3) = g'(x_2) \frac{f'(x_2 - x_3)h'(x_3)}{\sum_{x_1=-\infty}^{\infty} f'(x_1)h'(x_2 - x_1)}. \tag{15}$$

Taking the summation of both sides of Eq. (15) with respect to x_2 , using the relation of Eqs. (4), (5), and (6), and noting that

$$\sum_{x_2=-\infty}^{\infty} f'(x_2 - x_3) = 1, \tag{16}$$

we have the following relation.

$$h(x) = h(x) \sum_{x_2=-\infty}^{\infty} \frac{f(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} f(x_1)h(x_2 - x_1)}. \tag{17}$$

Thus, using the same recursive relation as Eq. (11), we have

$$h_{m+1}(x) = h_m(x) \sum_{x_2=-\infty}^{\infty} \frac{f_n(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} f_n(x_1)h_m(x_2 - x_1)}. \tag{18}$$

In order to check the convergences of the recursive relations given by Eqs. (11) and (18), the following relations are used.

$$\sum_{x_2=-\infty}^{\infty} \frac{h(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} h(x_2 - x_1)f_n(x_1)} = 1, \tag{19}$$

$$\sum_{x_2=-\infty}^{\infty} \frac{f(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} f(x_1)h(x_2 - x_1)} = 1. \tag{20}$$

$$\tag{21}$$

Thus, we use the following criteria to stop the iterations.

$$1 - \epsilon < \sum_{x_2=-\infty}^{\infty} \frac{h_m(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} h_m(x_2 - x_1)f_n(x_1)} < 1 + \epsilon, \tag{22}$$

$$1 - \epsilon < \sum_{x_2=-\infty}^{\infty} \frac{f_n(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} f_n(x_1)h_m(x_2 - x_1)} < 1 + \epsilon. \tag{23}$$

Using the above relations, Richardson has proposed the following iterative algorithm (Richardson's Iterative Method).

Step 1. Set $n = 0, m = 0$, the initial guesses of $h_0(x)$, and $f_0(x)$, and small positive number ϵ .

Step 2. Solve the following equations:

$$f_{n+1}(x) = f_n(x) \sum_{x_2=-\infty}^{\infty} \frac{h_m(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} h_m(x_2 - x_1)f_n(x_1)} \tag{24}$$

$$h_{m+1}(x) = h_m(x) \sum_{x_2=-\infty}^{\infty} \frac{f_n(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} f_n(x_1)h_m(x_2 - x_1)}. \tag{25}$$

Step 3. If the following inequalities hold

$$\left| \sum_{x_2=-\infty}^{\infty} \frac{h_m(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} h_m(x_2 - x_1)f_n(x_1)} \right| < 1 - \epsilon \tag{26}$$

and

$$\left| \sum_{x_2=-\infty}^{\infty} \frac{f_n(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} f_n(x_1)h_m(x_2 - x_1)} \right| < 1 - \epsilon, \tag{27}$$

then stop, otherwise $n \leftarrow n + 1, m \leftarrow m + 1$ go to Step 2.

The above iteration has no proof of convergence that means the results obtained by the above iteration may result in the good results or may not.

IV. PROPOSED ALGORITHM I

In order to get better results compared with Richardson's algorithm, we consider a new method based on the property of degraded images such that the blurred images are similar to the original images. In the Richardson's algorithm, if the bad estimation of $h_m(x)$ at the beginning stage, corresponding recovered images would become different images. After obtaining the bad estimation of recovered images, estimation of the point spread function turns worse. As a result, the iteration will produce worse and worse estimation of the point spread function and recovered images. Assuming the degraded images are not so far from the original images, we use the blurred image to estimate the point spread function $h_m(x)$ instead of the recovered image that is the estimated image. Therefore, we have proposed the following steps:

Algorithm I

Step 1. Set $n = 0, m = 0$, small positive number ϵ , and $f_0(x) = g(x)$. Set the initial guesses of $h_0(x)$.
 Step 2. Solve the following equations:

$$f_{n+1}(x) = f_n(x) \frac{\sum_{x_2=-\infty}^{\infty} \frac{h_m(x_2-x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} h_m(x_2-x_1)f_n(x_1)}}{\sum_{x_1=-\infty}^{\infty} h_m(x_2-x_1)f_n(x_1)} \quad (28)$$

$$h_{m+1}(x) = h_m(x) \frac{\sum_{x_2=-\infty}^{\infty} \frac{f_n(x_2-x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} g(x_1)h_m(x_2-x_1)}}{\sum_{x_1=-\infty}^{\infty} g(x_1)h_m(x_2-x_1)} \quad (29)$$

Step 3. If the following inequalities hold

$$\left| \frac{\sum_{x_2=-\infty}^{\infty} \frac{h_m(x_2-x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} h_m(x_2-x_1)f_n(x_1)}}{\sum_{x_1=-\infty}^{\infty} h_m(x_2-x_1)f_n(x_1)} \right| < 1 - \epsilon \quad (30)$$

and

$$\left| \frac{\sum_{x_2=-\infty}^{\infty} \frac{f_n(x_2-x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} f_n(x_1)h_m(x_2-x_1)}}{\sum_{x_1=-\infty}^{\infty} f_n(x_1)h_m(x_2-x_1)} \right| < 1 - \epsilon, \quad (31)$$

then stop, otherwise $n \leftarrow n + 1$ and go to Step 2.

V. VARIATION USING THE REVERSE FUNCTION

We consider a more simple form of the proposed algorithm. We set the denominator of Eq. (38) by

$$L_{nm}(x_2) = \sum_{x_1=-\infty}^{\infty} h_m(x_2-x_1)f_n(x_1). \quad (32)$$

It is the convolution sum between the original image $f_n(x_1)$ and the point spread function $h_m(x_2-x_1)$. Therefore, if $L_{nm}(x_2) = g(x_2)$, then the estimated image $f_n(x_1)$ becomes the true original image. Furthermore, we have

$$f_{n+1}(x) = f_n(x) \frac{\sum_{x_2=-\infty}^{\infty} \frac{h_m(x_2-x)g(x_2)}{L_{nm}(x_2)}}{L_{nm}(x_2)}. \quad (33)$$

We define $r_{nm}(x_2)$ by

$$r_{nm}(x_2) = \frac{g(x_2)}{L_{nm}(x_2)} \quad (34)$$

which means the ratio between the observed degraded image and the degraded image obtained by using the estimated point spread function. Then Eq. (33) becomes

$$f_{n+1}(x) = f_n(x) \sum_{x_2=-\infty}^{\infty} h_m(x_2-x)r_{nm}(x_2). \quad (35)$$

We define the reverse function of $k(x)$ by $k(x) = h(-x)$. Then Eq. (33) becomes

$$f_{n+1}(x) = f_n(x) \sum_{x_2=-\infty}^{\infty} \frac{k_m(x_2-x)g(x_2)}{L_{nm}(x_2)}. \quad (36)$$

If we represent the convolution sum by Fourier transform, we have

$$\begin{aligned} f_{n+1}(x) &= f_n(x)FT^{-1}(FT(k_m(x-x_2))FT(r_{nm}(x_2))) \\ &= f_n(x)FT^{-1}(K_m(j\omega)R_{nm}(j\omega)) \end{aligned} \quad (37)$$

where FT and FT^{-1} denote the Fourier transform and inverse Fourier transform. Since $K_m(j\omega) = H_m(-j\omega)$, we could save the computational time by half.

VI. SIMULATION RESULTS OF THE PROPOSED ALGORITHM I

In order to show the effectiveness of the proposed method, we will consider gray image (Example 1) and two of color images (Example 2, Example 3). The computer specification used here is shown in TABLE I. In Example 1 the gray image of 64×64 was made using Photoshop. The color images of 512×512 of Example 2 and Example 3 were cropped from the standard sample data of high-resolution color images [8]. The degraded images are made by using Gaussian filters with the standard deviation $\sigma = 3.3$. We used the stopping parameters of m and n when maximum iteration number k is given. In these simulations, we changed those parameters (m, n, k) as $(10, 100, 10)$, $(5, 100, 10)$, and $(5, 5, 100)$. TABLE II shows simulation results for three cases with PSNR (Peak Signal-to-Noise Ratio) in case of Example 2. Fig. 2 shows the simulation results of the gray image with $(10, 100, 10)$. From this Fig. 2 the proposed method restored a clearer image compared with the results of Richardson's method [5]. In Figs.3-10 we show the simulation results of Example 2 and Example 3 with images obtained from [8]. The original images are shown in Fig. 3 and Fig. 7. The degraded images are shown in Fig. 4 and Fig. 8. The restored images by [5] and the proposed method are shown in Fig. 5 and Fig. 9.

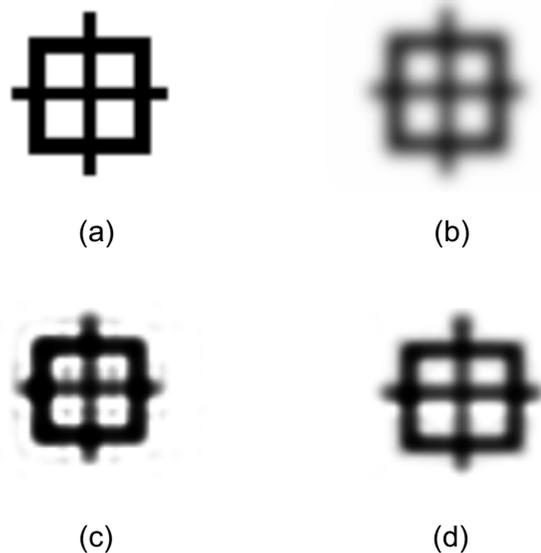


Fig. 2. The comparison for Example 1.



Fig. 3. Original image for Example 2: Cafeteria.

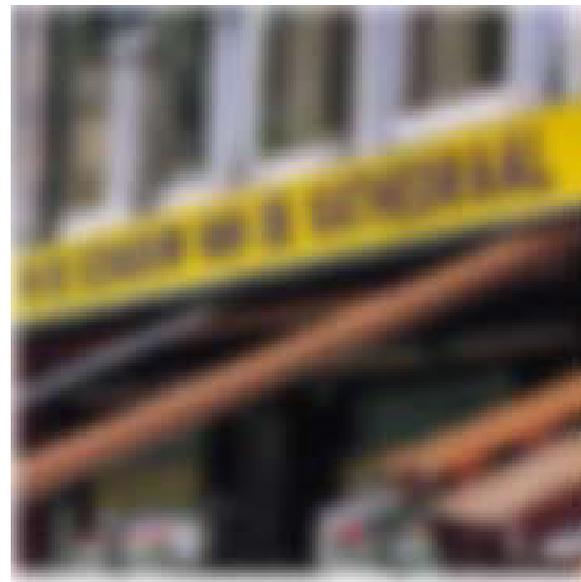


Fig. 4. Degraded image for Example 2.



Fig. 5. Richardson's method for Example 2.



Fig. 6. Proposed Algorithm I for Example 2.

TABLE I
COMPUTING ENVIRONMENT

OS	Windows XP
CPU	AMD Athlon(tm)64 X2 Dual Core
Memory	2GB

TABLE II
PSNR BETWEEN ORIGINAL IMAGE AND RESTORED IMAGE

Threshold value(m,n,k)	Degraded image	Richardson	Authors
(10,100,10)	14.5	15.7	16.6
(5,100,10)	14.5	16.5	16.0
(5,5,100)	14.5	9.2	16.2

VII. PROPOSED ALGORITHM II

The above algorithm has some problems. One is a convergence problem for iteration. We must select a suitable initial function of the point spread function to obtain good results.

The other problem is the speed of convergence. To solve these problems, we assume that the logarithmic amplitude spectrum of the point spread function is Gaussian distribution. This is



Fig. 7. Original image for Example 3: Fruits basket.



Fig. 8. Degraded image for Example 3.



Fig. 9. Richardson's image for Example 3.



Fig. 10. Proposed Algorithm I for Example 3.

based on the fact that the power spectrum of the point spread function is equi-uniform for each frequency. To speed up the computation to estimate the point spread function, we select a small area of the image where the low frequency seems to be dominant.

A. Estimation of the Point Spread Function

It is well-known that the spectrum of the point spread function includes the low frequencies compared with those of the original images. From Eq. (2) we can see that the spectra of the observed image are almost similar to the spectra of the point spread function. Thus, we can estimate the spectrum region of the point spread function from the blurred image (observed image). Since the major part of the low frequency region is affected by the zero frequency component, we adopt the logarithmic amplitude spectrum of the blurred image to remove the zero frequency. Thus, the computation

process to estimate the point spread function is given by the following steps.

Step 1. Applying the Hamming window to the blurred image, find the logarithmic amplitude spectrum of the obtained image.

Step 2. Normalize the logarithmic amplitude spectrum within $[0, 255]$ and threshold it with 128. The image is called a binary image.

Step 3. To remove the impulsive noise, apply 5×5 median filter to the binary image.

Step 4. Find the center of the binary image and determine the radius r by using the least mean squares method.

Step 5. Set $r = 3\sigma$ and find the Gaussian distribution function.

Step 6. Using inverse Fourier transform, find the point spread function $h(x)$.

To show the process stated above, we will use a sub-area



Fig. 11. The original image: Wine and tableware and sub-image to estimate the point spread function. Here, the sub-image is denoted by the square area and it is assumed to be known the point spread function with the spectrum as shown in the image.

of the original image with the known point spread function as shown in Fig. 11. Using this sub-image, we apply the procedure to an image given by Fig. 12.

B. Algorithm II

The estimate of the point spread function by using Step 1 to Step 6 is used as a rough estimate of the point spread function in order to converge the iteration algorithm in Algorithm I. Thus, Algorithm II is given by the following steps:

Step 1. Set $n = 0, m = 0$, small positive number ϵ . Furthermore, $f_0(x) = g(x)$ and $h_0(x) = h_p(x)$ where $h_p(x)$ denotes the estimated $h(x)$ in the above Section.

Step 2. Solve the following equations:

$$f_{n+1}(x) = f_n(x) \frac{\sum_{x_2=-\infty}^{\infty} \frac{h_m(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} h_m(x_2 - x_1)f_n(x_1)}}{\sum_{x_1=-\infty}^{\infty} h_m(x_2 - x_1)f_n(x_1)} \quad (38)$$

$$h_{m+1}(x) = h_m(x) \frac{\sum_{x_2=-\infty}^{\infty} \frac{f_n(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} g(x_1)h_m(x_2 - x_1)}}{\sum_{x_1=-\infty}^{\infty} g(x_1)h_m(x_2 - x_1)} \quad (39)$$

Step 3. If the following inequalities hold

$$\left| \frac{\sum_{x_2=-\infty}^{\infty} \frac{h_m(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} h_m(x_2 - x_1)f_n(x_1)}}{\sum_{x_1=-\infty}^{\infty} h_m(x_2 - x_1)f_n(x_1)} \right| < 1 - \epsilon \quad (40)$$

and

$$\left| \frac{\sum_{x_2=-\infty}^{\infty} \frac{f_n(x_2 - x)g(x_2)}{\sum_{x_1=-\infty}^{\infty} f_n(x_1)h_m(x_2 - x_1)}}{\sum_{x_1=-\infty}^{\infty} f_n(x_1)h_m(x_2 - x_1)} \right| < 1 - \epsilon, \quad (41)$$

then stop, otherwise $n \leftarrow n + 1$ and go to Step 2.

TABLE III
COMPUTING TIME

	Algorithm I	Algorithm II
Time [s]	38179.56	846.95

C. Speed-Up of Algorithm II

In order to speed up the algorithm, we should use the small size training image as shown in Fig. 11 instead of the full size image. the selection of the sub-image is the most important to obtain a better estimation of the point spread function. Since we can assume that the point spread function is the same even if the image sizes are different, we use a small size image instead of a large size image. We use images with 2560×2048 and select 256×256 to estimate the point spread function. The problem is where we should select the sub-images for training the point spread function. Estimation of Bayesian principle requires the broad band input data as in the system parameter identification to get the high performance. Thus, we select the high frequency images which include many edges. To find the edges, we use the Laplacian filter and take the area with large variance. Therefore, we select the sub-image area by the following steps:

Step 1. Apply the Laplacian filter to the observed image.

Step 2. Select the sub-image with 256×256 .

Step 3. Find an area with the largest variance from the above sub-images.

By the above steps we will select the training area to estimate the point spread function stated in Section VII-A.

VIII. SIMULATION RESULTS

We consider Cafeteria, Fruits and basket, and Wine and tableware whose true images are known in advance. After applying Algorithm I and Algorithm II to those images, we check the case where the true image is not obtained and the blurred images are taken by using a high resolution camera without adjusting so precisely which results in a little bit blurred images. We show the simulation results for the former case in Figs. 13-15 which are corresponding to Figs. 2, 7, and 11, respectively.

Compared with the restored images by Algorithm I, the restored images seem to be similar to the original ones. Furthermore, the computational time by Algorithm II is faster than that of Algorithm I as shown in Table III.

Finally, we have applied Algorithm I and Algorithm II to the blurred images which are taken by using a high resolution camera. Here, we took artificially blurred images without auto-focus operation. Restored images are shown in Figs. 16 and 17. The case of high density camera makes clear images when the image scales are small. But if we enlarge them, then the difference become clear as shown in Fig. 17. From this result, the Algorithm II could be used to restore the blurred images.

IX. CONCLUSIONS

In this paper, methods of restoration of the degraded images by using the Bayesian-based iterative method are proposed.

The simulation results showed that the proposed method could restore the degraded images more clearly compared with the Richardson's method while the threshold values of (n, m, k) must be determined by trial and error.

Furthermore, the computation load has been decreased by half by introducing the ratio between the observed degraded image and the degraded image by using Algorithm I. Algorithm II could restore the blurred images more precisely and faster compared to Algorithm I since the latter method has only selected the initial image as the observed image without considering the initial guess of the point spread function.

Algorithm II takes more time to estimate the point spread function. But taking into consideration that the point spread function does not vary from place to place in the same image, we can select small region to estimate the function. In this paper, we select the sub-image where the variation of the image is rather high, that is, the image includes many edges. Since we took the sub-image with 256×256 size, it did not require so much time.

An open problem is that the Bayesian approach might not have a solution since the convergence of the iterative method has not been proved. But the initial guess may not be so far from the solution of the Bayesian equation since the blurred image structure will keep the similar behavior to the original images.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant-in-Aid for Scientific Research (B) (23360175). The authors would like to thank JSPS to support this research work.

REFERENCES

- [1] S. Omatu and H. Araki, Image Restoration by Revised Bayesian-Based Iterative Method, ADVCOMP 2011, Lisbon, Portugal, (2011)
- [2] T. Young and K. Fu, Handbook of Pattern Recognition and Image Processing Independent Component, Academic Press, New York, (1986)
- [3] R. Duda, P. Hart, and D. Stork, Pattern Classification, John Wiley & Sons, New York, (2001)
- [4] Y. Yizhaky, I. More, A. Lantzman, and N. S. Kopeika, Direct Method for Restoration of Motion-Blurred Images, Journal of Optical Society of America, Vol. 15, pp. 1512-1519, (1998)
- [5] W. Richardson, Bayesian-Based Iterative Method of Image Restoration, Journal of Optical Society of America, Vol. 62, pp.55-59, (1972)
- [6] B. Lucy, An Iterative Method for the Rectification of Observed Distributions, JASTON. Journal, Vol. 79, pp. 745-754, (1974)
- [7] R. G. Lane, Methods for Maximum-Likelihood Deconvolution, Journal of Optical Society of America, A, Vol. 13, pp. 1992-1998, (1972)
- [8] Standard Color Digital Images of High-Resolution(CMYK/SCID), JIS X 9201:2001, (2001)

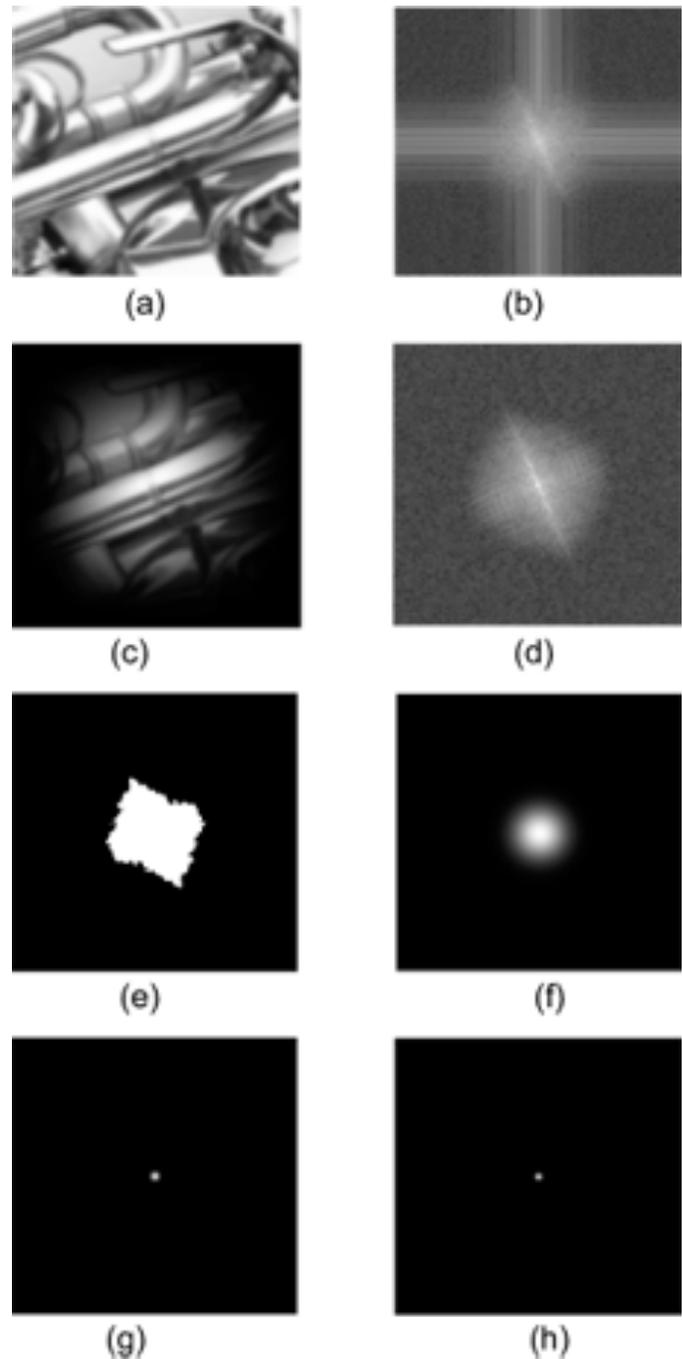


Fig. 12. The estimation process of the point spread function. Here, (a) is the blurred image denoted in Fig. 11, (b) is the logarithmic amplitude spectrum of (a), (c) is the blurred image filtered by the hamming window, (d) is the logarithmic amplitude spectrum of (c), (e) is the estimated logarithmic amplitude spectrum of (d) after processing of Step 2 and Step 3, (f) is estimated Gaussian distribution of (e) after processing Step 4 and Step 5, (g) is the estimated point spread function, and (h) is the true point spread function.

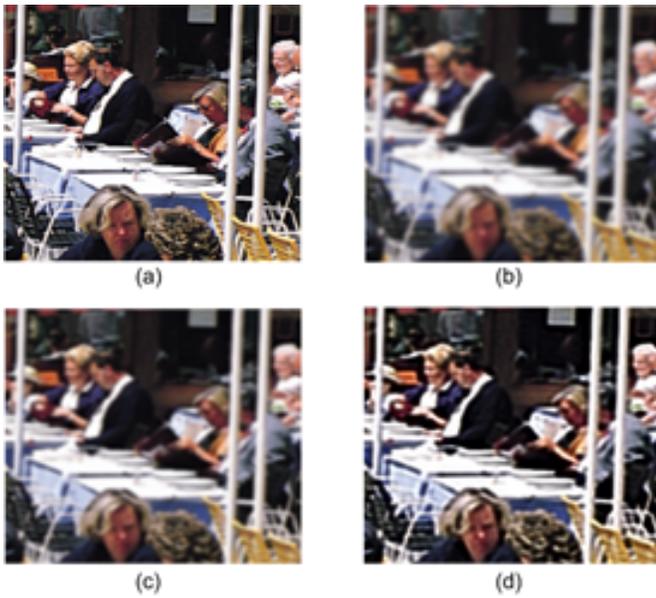


Fig. 13. The simulation results for the image: Cafeteria. Here, (a) is the original image, (b) is the blurred image, and (c) is the image by Algorithm I, and (d) is the image by Algorithm II.

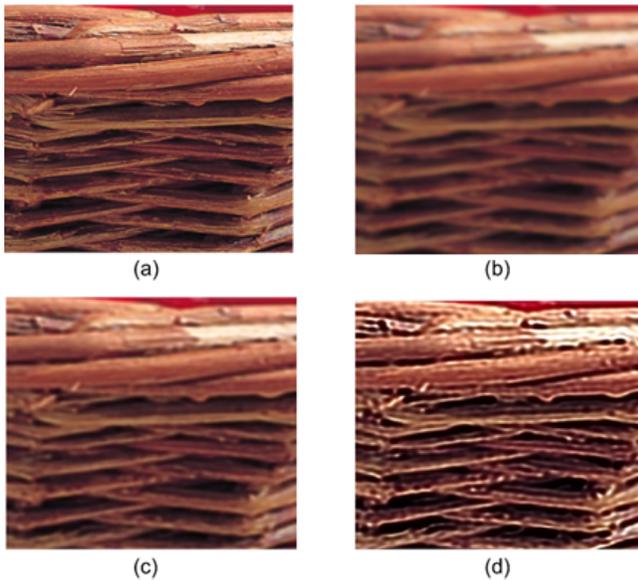


Fig. 14. The simulation results for the image: Fruits basket. Here, (a) is the original image, (b) is the blurred image, and (c) is the image by Algorithm I, and (d) is the image by Algorithm II.

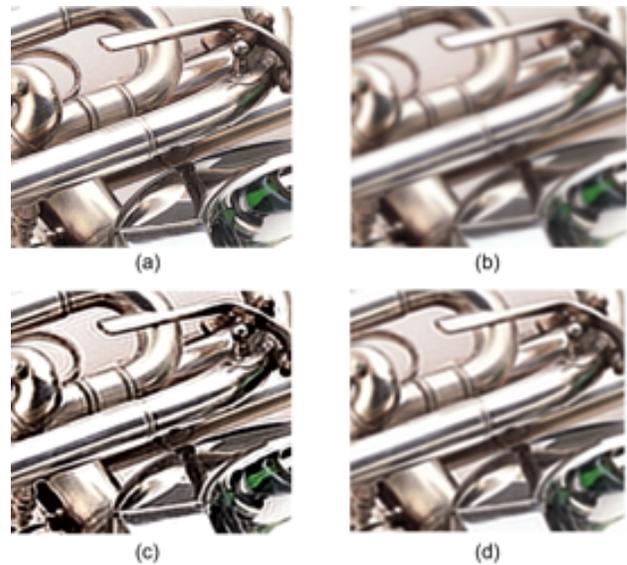


Fig. 15. The simulation results for the image: Wine and tableware. Here, (a) is the original image, (b) is the blurred image, and (c) is the image by Algorithm I, and (d) is the image by Algorithm II. known point spread function with the spectrum as shown in the image.



(a)



(b)



(c)

Fig. 16. The simulation results for the blurred image: Osaka Prefecture University. (a) is the blurred image, (b) is the image by Argorithm I, and (c) is the image by Argorithm II. known point spread function with the spectrum as shown in the image.



(a) Blurred image



(b) Algorithm I



(c) Algorithm II

Fig. 17. The comparison of the small area. (a) is the blurred image, (b) is the image by Argorithm I, and (c) is the image by Argorithm II.

Energy-aware MPSoC for Real-time Applications with Space-Sharing, Adaptive and Selective Clocking and Software-first Design

Stefan Aust and Harald Richter
 Dept. of Computer Science
 Clausthal University of Technology
 Clausthal-Zellerfeld, Germany
 stefan.aust\harald.richter@tu-clausthal.de

Abstract—Energy-awareness is an important criterion for many mobile appliances such as (smart)phones and handhelds. It is also indispensable for electronic controller units in cars for example. Unfortunately, low energy consumption and high-computing power exclude each other. With the proposed methods of space-sharing, adaptive and selective clocking and software-first design, both goals can be reached simultaneously. Space-sharing is an alternative to time-sharing for multi-task controllers in real-time systems that significantly simplifies task scheduling. With space-sharing, there is no need for worst-case execution-time analysis. Furthermore, adaptive and selective clocking, together with a software-first design reduce the controller's energy consumption to the absolute minimum. The results described herein were achieved by a set of measurements made at a single-chip multiprocessor system called MPSoC1 that implements space-sharing on one FPGA and by a second system in software-first design called MPSoC2 that implements adaptive and selective clocking.

Keywords—real-time system; low power; multiprocessor system on chip (MPSoC); worst-case execution time (WCET); space-sharing; adaptive and selective clocking; software-first design.

I. INTRODUCTION

A MPSoC is a parallel computer on a single silicon chip that may contain between two and several hundreds of processing units. In the case of up to ten units, we may speak of a multi-core processor, otherwise of a many-core CPU. Typically, a multi-core processor employs an on-chip first-level cache that is shared between all cores for interprocess communication. If more cores than about 10 are present on the same chip, shared-memory can not be used any more because of the memories bandwidth saturation and other communication means have to be used. Many-core CPU have therefore an on-chip static or dynamic interconnection network for interprocess communication that is normally not real-time capable.

Furthermore, multi- and many-core CPUs are usually implemented by a full-custom-design chip. But during the last years, the capabilities of Field Programmable Gate Arrays (FPGAs) have been increased so much that they are in many cases an alternative to full-custom chips. Every state-of-the-art FPGA can accommodate already now hundreds of processing units, i.e., cores, together with a static or dynamic interconnection network. However, the disadvantage of such

FPGA solutions is that each core is less powerful as in the full-custom design due to low clockrates and that less memory is available on chip. This can be overcome by a higher number of cores and by software that is coded in the parallel programming style, together with a proper intertask synchronization.

In daily life, there are many mobile appliances that demand high computing power, but have low energy resources only. This creates a contradiction because high computing power normally means high energy consumption as well. Additionally, precious energy must be invested to cool-down these devices. As a consequence, either the usability or the operational time is short. An other fact is that thermal dissipation limits the life expectancy of electronic devices because of aging processes in the semiconductor material. The pn-junctions in the transistors deteriorate with increasing heat exposure. Finally, heat dissipation always means low energy efficiency, which is the opposite of Green IT. Because of these issues, energy-awareness is important for mobile systems and for ECUs in cars as well.

The concept of space-sharing was introduced first by the authors in [1]. Based on space-sharing we suggested in [2] the usage of MPSoCs on a single FPGA to provide high computing power for energy-limited real-time applications. Space-sharing instead of time-sharing eliminates the problem of finding and guaranteeing a proper time schedule for multiple tasks that are needed to meet the prescribed functionality under a given set of time constraints. Furthermore, it allows also a better energy efficiency in embedded systems if combined with adaptive and selective clocking and a so-called software-first design because this will reduce dynamic power dissipation. As a result, space-sharing is able to reduce energy consumption and to produce high computing power.

The paper is organized as follows: In section II, an overview of the state-of-the-art of time-sharing and space-sharing is given. In section III, the architecture of the MPSoC1 that implements space-sharing in one FPGA is presented. In section IV, measurement results of the energy consumption of MPSoC1 are given. In section V, MPSoC2 is presented, together with methods for lower energy con-

sumption. In section VI, MPSoC1 is compared to MPSoC2 with the same user application that illustrates the benefits of the proposed methods. Section VII draws a conclusion of the achieved results and gives an outlook to future work.

II. STATE-OF-THE-ART OF TIME-SHARING AND SPACE-SHARING

Real-time applications demand the time-sliced execution of a sequence of tasks on a single processing unit under the boundary condition of given time limits or at given points in time. A common method to evaluate the soft or hard real-time character of an application is the so-called worst-case execution-time (WCET) analysis. This analysis is made for the processing unit that executes that tasks by calculating how long each task will need to complete in the worst case [3].

In embedded systems with time-sharing operation, delays in task execution can occur that have several reasons, such as 1.) matching the deadlines of all tasks by one processor, which forces concurrency, i.e., competition between tasks, 2.) exchanging intermediate results between tasks because of interprocess communication, and 3.) interruption of one task by an other task of higher priority. In order to avoid intolerable delays in task execution, a suitable scheduling must be found that guarantees desired operation under all circumstances. Several task scheduling strategies have been created to solve this problem. These are mainly priority scheduling, earliest deadline first or round robin [4]. Furthermore, WCET analysis grows exponentially with the number of tasks because the amount of if-then-else branches span up a tree of execution paths, which must be all traversed to find the longest path. Because of the fact that WCET is a NP-complete problem, many commercial and open-source analysis-tools have been developed to ease WCET analysis [5], [6]. With space-sharing, there is no need for task scheduling and thus also not for WCET analysis.

In space-sharing, every task gets its own processing element, which is a core or a whole CPU that is part of an on-chip parallel computer [1]. Additionally, every interrupt service routine and every device driver gets its own processing element as well. Thus it does not happen that one task is interrupted by an other. This means that the number of processing elements must match the cumulative number of all tasks (Fig. 1).

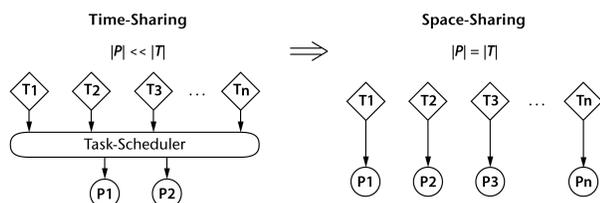


Figure 1. Time-sharing vs. space-sharing.

Furthermore, by this means every task gets its own local memory where it resides with program and data. As a consequence, tasks are never competing for the same resources. However, in case of interprocess communication the execution of a task can still be delayed if the task must wait until a corresponding task has calculated a required intermediate result. This problem is known as task synchronization. It can not be solved by WCET analysis but by proper parallel programming.

The combination of storage and computing element is called processor-memory-module (PMM). Several or many PMMs are coupled by a static or dynamic interconnection network that is on the same chip and real-time capable. Finally, PMMs can be connected to peripheral devices such as sensors, actuators, hard disk, network interface or external memory by the chip's IO pins.

Space-sharing requires that the FPGA has enough resources to accommodate all needed components, and it requires that practical methods exist for allocating tasks to processors, as well as for automatic chip synthesis due to the number and structure of PMMs. It requires also that simple means exist to compile code and to debug it for every PMM, and that intertask communication occurs in real-time.

III. ARCHITECTURE OF MPSOC1

Fig. 2 shows the architecture of MPSoC1 that we have implemented on various FPGAs from Xilinx. MPSoC1 consists of a configurable number of PMMs and memory sizes and a multistage interconnection network that has switches of size 2×2 . Each PMM can be directly coupled to an I/O pin or can access peripheral resources via the interconnection network. Because of the spatial isolation of PMMs, local program code and data are protected from unintended overwriting by other tasks. This means that interprocess communication is either possible by passing messages through the network or by shared variables that reside in a global (external) memory. Furthermore, we managed to develop a VHDL program that synthesizes a real-time capable interconnection network of configurable size, together with an arbitrary number of PMMs. Furthermore, there exist simple procedure calls for intertask communication and for clock-rate setting.

The processing elements in our implementation are soft-core processors. For these processors, compilers and debuggers exist, but for each PMM code must be compiled and debugged separately. Open issues are therefore by which software tools programming and debugging of the parallel computer can be accomplished as a whole, how tasks are allocated automatically by a middleware to PMMs, what operating system can be run on the FPGA, and whether the number of logic cells on the FPGA is sufficient to accommodate all tasks. These problems are especially imminent as soon as very complex real-time applications should be solved by space-sharing or as soon as complex device drivers are needed to access peripherals. For example, the

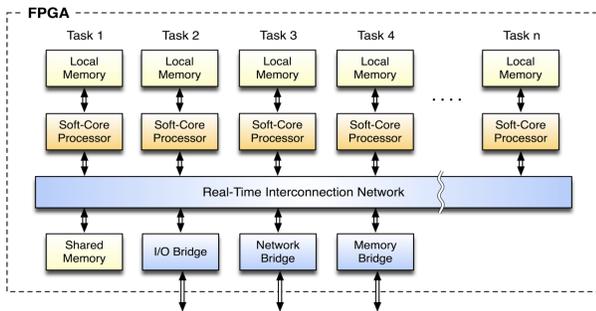


Figure 2. MPSoC architecture on a FPGA.

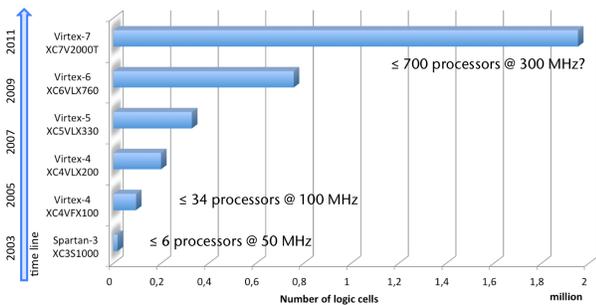


Figure 3. Evolution of Xilinx FPGAs within the last few years.

ECU software in modern cars comprises hundreds of tasks and thousands of intertask communications between ECUs. The questions arising from that issues will be discussed in the next subchapters. It will be shown that space-sharing is a usable alternative to time-sharing for many real-time applications.

A. FPGA Evolution

As shown in Fig. 3, FPGAs have advanced significantly with respect to the number of logic cells and internal memory within the last few years, and this trend will continue. For example, a Virtex-4 XC4VFX100 FPGA has less than 5 percent of the capacity of the latest Virtex-7 XC7V2000T FPGA. The same holds for on-chip memory, which has reached up to 70 Mbits per chip. In consequence, a Virtex-7 FPGA from Xilinx for instance is able to host hundreds of PMMs, albeit with a much lower processor clock rate compared to full-custom designed processors and with smaller local memories. However, a clock of 200 MHz and a program and data store of several dozens of KByte is feasible for a Virtex-7 FPGA for about 2-300 PMMs, which is sufficient for many feed forward and feed back control algorithms. Also single-chip controllers have no higher clock frequencies. The reason for that is the necessity for GHz and GBytes in embedded systems is low.

B. Soft-Core Processors

A soft-core processor exists as a set of FPGA logic cells that are synthesized via a hardware description language

such as VHDL or Verilog. In our tests, we have used a publicly available processor description from Xilinx called MicroBlaze [7]. On a Virtex-5 PFPGA, for example, each single MicroBlaze consumes 2-3 % of the chip's logic cells and memory. A MicroBlaze implements an in-order, non-superscalar, 32-bit RISC CPU with a clock rate of up to 200 MHz. Because of its simple RISC architecture, it is possible to predict the CPU cycles needed for a given real-time task more easily than in the case of a fully-featured CPU. MicroBlazes can even be synthesized without caches and branch prediction, which allows to calculate exactly the execution time for every task. Application software can be developed in C or C++ because compilers are existing in the Xilinx EDK toolset, from other vendors and open sources as well.

C. Interconnection Network

For interprocessor communication we used a multistage interconnection on-chip network (MINoC), which is based on the Beneš-network and which establishes real time communication paths between PMMs. Messages are transferred in a point-to-point manner or as multicast or broadcast. All three communication types are realized by $(2N \cdot \log_2 N - 1)$ 2×2 switches that can transfer data either as a through ("=") or as a crossed ("x") connection or as broadcast from one switch input to both outputs. Each input port to the interconnection network is equipped with a message FIFO to decouple message creation in a PMM from message transfer and delivery in the network. The FIFOs store the messages that are destined for a specific output port as long as that port is occupied. The FIFO depth is small in order not to impair real-time capability that would be caused by messages that are waiting too long in a FIFO. Since the interconnection network works in circuit switching mode, a direct path through the network from sender to receiver is established for every communication type as long as the communication persists. The network can connect every input with every output at any time, as long as no two inputs want to be transferred simultaneously to the same output port. Such a situation is considered as bad task synchronization.

The most important feature of the interconnection network is its non-blocking character, which is a consequence of its topology. This feature comes from the fact that alternative paths can be switched through the network during run-time. The network routing algorithm was developed by one of the authors [8], [9] and [10]. Later the routing was improved by the other author [11]. The improved routing algorithm is able route N paths from inputs to outputs within one clock cycle [12] by means of combinatorial logic that is implemented in AND, OR and NOT gates. The non-blocking character of the network is mandatory for real-time communication. Otherwise connection requests would be delayed because of network-internal conflicts, and no upper time limit could be guaranteed for message latency.

D. Design Methodology and Operating System

In controllers with time-sharing, tasks are sequenced by a task scheduler that is part of a real-time operating system (RTOS). The RTOS provides additionally for device drivers, memory protection, interrupt handling and interprocess communication. However, most of these RTOS functions are obsolete in space-sharing. The only task per PMM can be executed in stand-alone mode if some prerequisites are fulfilled. These are: 1.) a communication library is provided that implements point-to-point, multicast and broadcast for message passing via the interconnection network, 2.) semaphores for shared memory access via a global memory are existing. 3.) Reading and writing sensors and actuators by the PMM is performed via specific device driver tasks in dedicated PMMs. The latter is possible as long as no complex devices such as graphic, hard disk or network controllers must be read and written. In that case, either a RTOS kernel must be employed additionally that has stripped-off all unnecessary features because of memory limitations in the PMM, or external memory that is on the FPGA board must be engaged.

The design methodology for application code in space-sharing is similar to that of parallel programming: first the application must be partitioned into several tasks. Then the needed interprocess communication must be defined, and code and communication libraries must be bounded together. Finally, the parallel program must be tested. In contrast to parallel programming, the number of PMMs, the structure of every PMM and the interconnection network must be synthesized for the target FPGA in space-sharing before program test. Furthermore, space-sharing allows to resize the local memory to the requirements of each task. Since soft processors are used, the processor architecture can be adapted to software requirements as well, for example by an additional coprocessor as hardware accelerator. Xilinx EDK allows to configure in detail every MicroBlaze as needed. Because of that, computing hardware depends on the application software, which we call software-first design.

E. Energy-awareness in Space-Sharing

According to [13], the total power consumption P of a semiconductor chip consists of static power dissipation P_{stat} and dynamic power dissipation P_{dyn} and it must hold:

$$P = \sum P_{stat} + \sum P_{dyn} \quad (1)$$

In the following, it will be discussed how the dynamic fraction of the FPGA power consumption can be reduced. The static power cannot be altered by chip users because it is caused by leakage currents inside of the semiconductor [14]. Dynamic power dissipation, however, arises from switching activities of transistors. Equation 2 determines the dynamic power dissipation as a function of supply voltage V , clock frequency f and chip capacitance C [15].

$$P_{dyn} = \sum C \cdot V^2 \cdot f \quad (2)$$

According to Eq. 2, there are three options for reducing dynamic power dissipation:

- lower switching capacitance C
- lower supply voltage V
- lower switching frequency f

The switching capacitance depends on the production process of the chip and cannot be controlled by FPGA users. The supply voltage can be controlled by an adaptive power supply for the whole chip. However, the switching frequency can be controlled by an adaptive clock for every PMM. Such clocks can be implemented by a central clock generator for the chip and by individual clock dividers at every PMM. If the divided output clock can be disabled then the PMM can also be stopped and restarted arbitrarily because no DRAM cells are inside of a FPGA that must be refreshed. Furthermore, only that area on the FPGA-chip must be clocked at all that is needed for space-sharing. The rest of the chip will dissipate only static power, which is much less. Finally, because of the fact that every PMM has its own clock the PMM's dynamic power dissipation can statically be reduced until the clock has reached the lowest possible periodicity that the user application allows. Additionally, dynamic power dissipation can be reduced adaptively if the clock divider is controlled by the PMM's task. Phases with high computing requirements are clocked faster by the task itself than phases with low requirements or with slow reaction times. The task knows when these phases occur because it is programmed by the user.

Finally, as long as the PMM waits for input from a peripheral device the clock can be switched off totally and restarted again by that peripheral when data are delivered. Such power saving potential is not available for embedded systems with time-sharing because their clock rates do not depend on the tasks. It would be too risky for high-speed tasks if the clock rate would be decreased by time-shared low speed tasks. Furthermore, if the clock is switched-off in time-sharing systems all tasks must rest for ever because an individual switch-off and easy hardware restart by a peripheral is not possible. Thus power dissipation remains in general the same in time-sharing systems unless the supply voltage is reduced. The regulating of the supply voltage is practiced in every laptop, for example. However, space-sharing can adopt voltage regulation as well. In Fig. 4, individual clocking is shown together with voltage regulation.

IV. MEASUREMENT RESULTS OF THE ENERGY CONSUMPTION OF MPSOC1

A. Test Setup

During our tests, we used three commercial FPGA evaluation boards with Xilinx Spartan-3, Virtex-4 and Virtex-5 FPGAs, which are listed in Table I. All boards are equipped

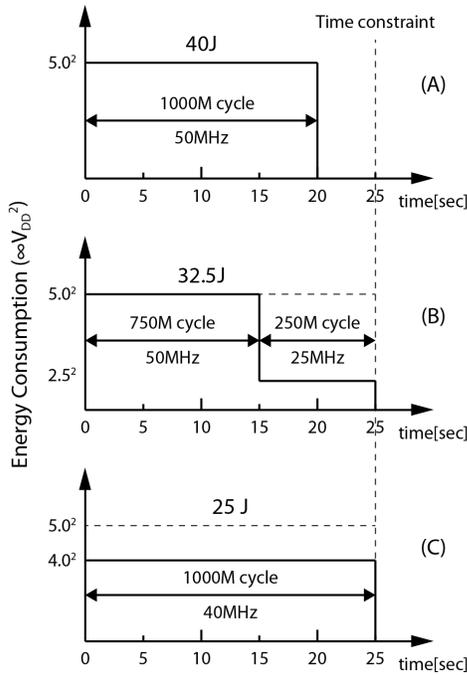


Figure 4. Motivational Example [16].

Table I
LIST OF TESTED FPGAS

FPGA	FPGA	
	Chip	Board
Spartan-3	Xilinx XC3S1000	Digilent Starter-Kit
Virtex-4	Xilinx XC4VFX100	PLDA XpressFX
Virtex-5	Xilinx XC5VLX50T	Xilinx ML505

with DIP switches that allow to control a clock divider for the FPGA. This enabled us to measure at different clock speeds without re-synthesizing the FPGA, which is important because a new synthesis would influence the result of the measurements as soon as components were placed and routed differently (cf. IV-F). In addition to that, the system clock had to be switched-off to measure static power dissipation, which was also possible by these DIP switches.

All boards have an external power supply where we could connect our test circuit as shown in Fig. 5. We used for all measurements the same adjustable power supply to avoid inaccuracies, together with two multimeters for voltage and current measurements. Each time we measured with and without clocking to separate dynamic from static power dissipation (cf. Eq. 1) .

B. Static vs. Dynamic Power Dissipation

On a Virtex-4 XC4VFX100 FPGA we could accommodate between 1 and 34 PMMs, consisting of a standard soft processors of type MicroBlaze and 16 KB memory each. The chip is manufactured in 90 nm gate size. At first all

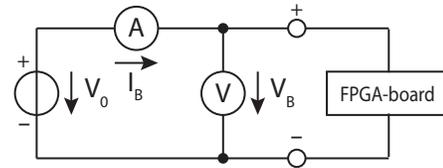


Figure 5. Test circuit for power measurements

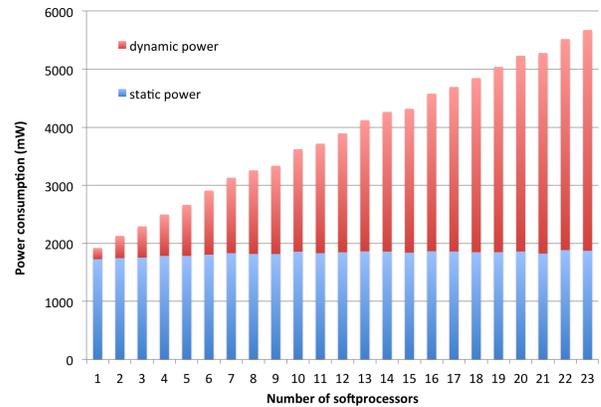


Figure 6. Static vs. dynamic power consumption for various numbers of soft processors, measured on Xilinx Virtex-4 FPGA.

processors were clocked with 100 MHz, while the total power consumption of the FPGA was measured. After this, the system clock was disconnected to measure the static power consumption only. Dynamic power dissipation was obtained as the difference of both measurements. The results of up to 23 soft processors are shown in Fig. 6, where blue columns indicate static power dissipation and red columns indicate dynamic power dissipation.

C. Influence of the Number of PMMs

Fig. 6 shows that static power dissipation remains constant while dynamic power dissipation increases linearly with the number of PMMs. Deviations from a straight line are caused by the place-and-route function of the synthesis tool, which was Xilinx XST [17]. Fig. 6 indicates also that for more than 12 PMMs dynamic power dissipation dominates. Other FPGAs show the same principal behavior as it can be seen from 7.

The absolute numbers we have measured in Fig. 7 are: a Spartan-3 PMM consumes 268 mW of dynamic power, a Virtex-4 PMM consumes 120 mW, and a Virtex-5 PMM consumes 53 mW.

D. Influence of the Processor Clock Rate

Another measurement series was conducted to get the dynamic power dissipation versus the processor clock rate. Various processor clock rates were investigated by using a

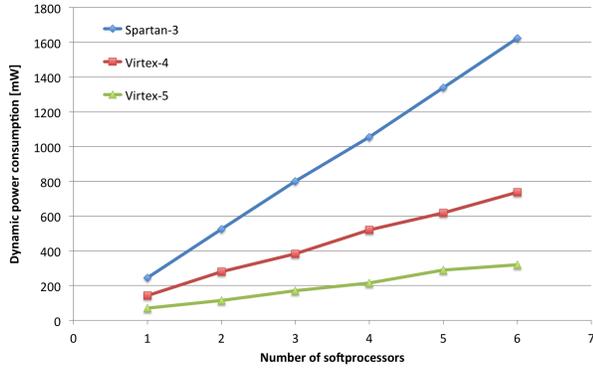


Figure 7. Dynamic power consumption for various numbers of soft processors, measured on Xilinx Spartan-3, Virtex-4, and Virtex-5 FPGAs.

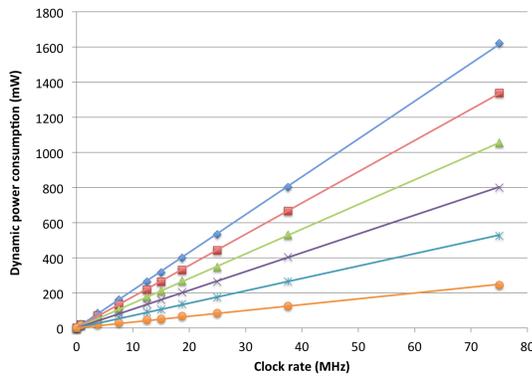


Figure 8. Dynamic power consumption vs. processor clock rate measured on Spartan-3 FPGA. The number of processors is parameterized.

clock divider while the number of PMMs was kept constant. In Fig. 8, the results of MPSoCs with 1 to 6 processors in a Spartan-3 FPGA are shown. In Fig. 9, the results of MPSoCs with 1 to 8 processors in a Virtex-5 FPGA are shown. All results comply with Eq. 2. Both FPGAs show a linear curve from which one can derive also the average dynamic power dissipation in absolute numbers according to (Eq. 3 and Eq. 4).

as:

Spartan-3 FPGA:

$$P_{dyn} = 3.64 \frac{mW}{MHz} \quad (3)$$

Virtex-5 FPGA:

$$P_{dyn} = 0.48 \frac{mW}{MHz} \quad (4)$$

E. Influence of the Manufacturing Technology

In all measurements of Fig. 7, the same soft processor architecture plus the same size of local memory were used. Nevertheless, the dynamic power dissipation varies with

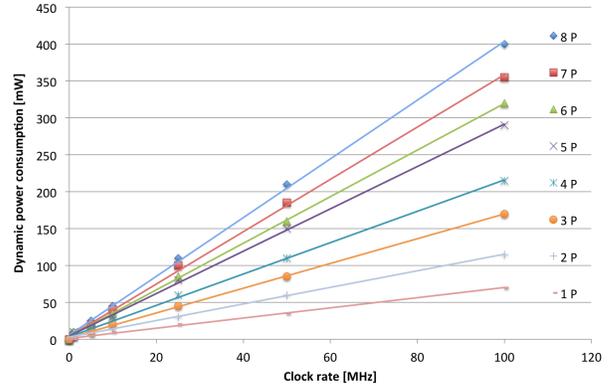


Figure 9. Dynamic power consumption vs. processor clock rate measured on Virtex-5 FPGA. The number of processors is parameterized.

different FPGAs. As one can see, the decrease of dynamic power consumption from one FPGA generation to the next is caused by smaller transistor sizes (65 nm instead of 90 nm) and other technological improvements.

F. Influence of the Place-and-Route Tool

The place-and-route tool is responsible to find space on the chip for all components and to connect them. The length of FPGA-internal connections and their switching capacitance vary from layout to layout. Thus place-and-route tools influence the dynamic power dissipation of FPGA-based systems. An analysis of this influence was made for example by Coxon in [18]. He showed that the dynamic power dissipation can be reduced up to 14% in Spartan-3, up to 11% in Virtex-4 and up to 12% in Virtex-5 FPGAs by optimizing the place-and-route process via user intervention that was made by synthesis directives. In Fig. 9, it can be seen that the distance between neighbor curves is not a constant for all curves. For example, the MPSoC with 5 processors consumes a little bit more power than expected. We explain this by the influence of the place-and-route tool.

G. Influence of the Processor Structure

We investigated the influence of the processor structure on the dynamic power dissipation by means of a Spartan-3 FPGA that was operated a clock rate of 50 MHz. The internal structure of a MicroBlaze can be configured in the Xilinx EDK toolset. The result is shown in Table II. A MicroBlaze that has a simple structure with few internal components only dissipates 182 mW dynamic power. Additionally 107 mW are used for a 5-stage pipeline and 119 mW for a floating point unit, for example.

Fig. 10 shows the dynamic power dissipation of a fully-equipped MicroBlaze. The percentage every component contributes in that processor to the total power is listed in Table II. From this table it can be seen that it pays out a lot to remove unused processor components, what in space-sharing depends of the application software. This result is

Table II
DYNAMIC POWER CONSUMPTION OF SOFT PROCESSOR COMPONENTS
MEASURED ON SPARTAN-3 FPGA

Processor Setup	Dynamic Power Dissipation
basic	182 mW
+ 5-stage pipeline	+ 107 mW
+ barrel shifter	+ 13 mW
+ 32-bit multiplier	+ 11 mW
+ integer divider	+ 15 mW
+ floating point unit	+ 119 mW

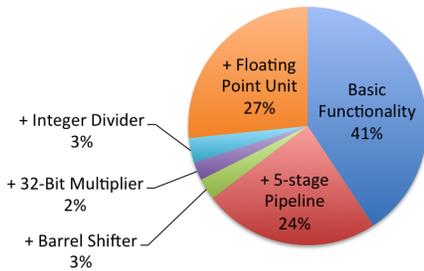


Figure 10. Percentage of dynamic power consumption of processor components in a fully equipped MicroBlaze soft processor as measured on Spartan-3 FPGA.

fully compliant with the software-first design methodology that was mentioned before.

H. Influence of the Local Memory

In order to investigate the influence of the local memory on dynamic power dissipation we measured several MPSoCs with different local memory sizes on a Virtex-4 FPGA at a clock rate of 100 MHz. The number of processors was parameterized. Each PMM executed the same test software that accessed local memory so that it was used by switching bits. Fig. 11 shows the result.

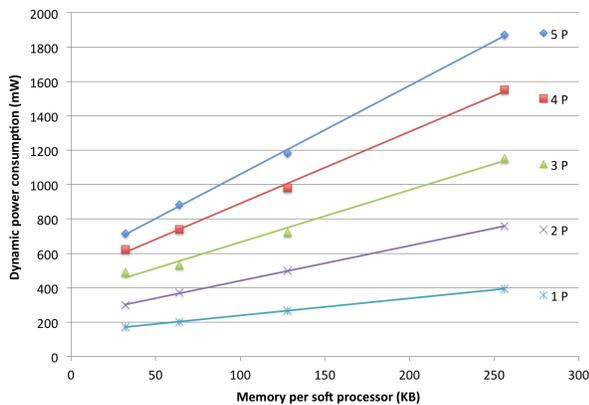


Figure 11. Dynamic power consumption vs. size of local memory measured on Virtex-4 FPGA. The number of processors is parameterized.

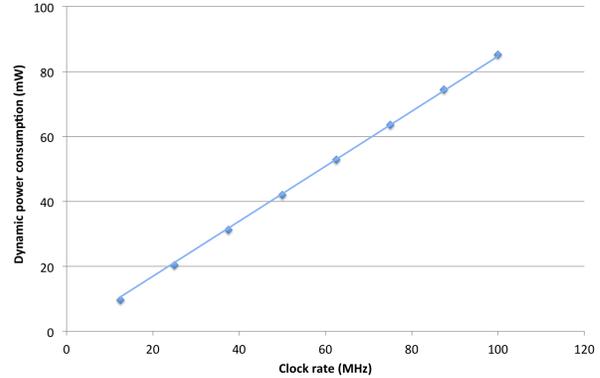


Figure 12. Dynamic power consumption of the MPSoC interconnection network vs. its clock rate measured on Virtex-4 FPGA.

One can also see in Fig. 11 by extrapolating the measurement curve to zero is that a PMM that has no memory consumes about 100 mW. Furthermore, the slope of the curve is about 1 mW per KB of memory. By combining the results from section IV-D with Fig. 11 we can establish to following empiric equation for the Virtex-4 FPGA:

$$P_{dyn} = 1 \frac{mW}{MHz} + 0.01 \frac{mW}{MHz \cdot KB} \quad (5)$$

I. Influence of the Interconnection Network

In Fig. 12, the influence of the MPSoC network clock rate on the dynamic power dissipation is presented for a Virtex-4 FPGA. The diagram shows that the network’s dissipation increases linearly with a slope of about 0.85 mW per MHz. Furthermore, we found that most of the dynamic dissipation mainly arises from the FIFO buffers that decouple processors at the network interfaces for asynchronous interprocessor communication. The network itself is very power efficient because its principle of circuit switching, i.e., signal transfer without buffering. In consequence, the less the FIFO buffer depth is the less dynamic power is consumed. However, the FIFO depth can not be chosen arbitrarily small because it depends on the number of messages that have to be temporarily stored, their size and how big the difference between processor clock rate and network clock rate is. A big speed difference requires a deep FIFO to balance-out message sending and transferring, at least for a while.

J. Power Consumption Constants

Overall, from the accomplished measurements we got the following numeric constants for dynamic power dissipation of a MPSoC that was implemented on a Virtex-4:

$$P_{processor} \approx 1 \frac{mW}{MHz} \quad (6)$$

$$P_{memory} \approx 0.01 \frac{mW}{KB \cdot MHz} \quad (7)$$

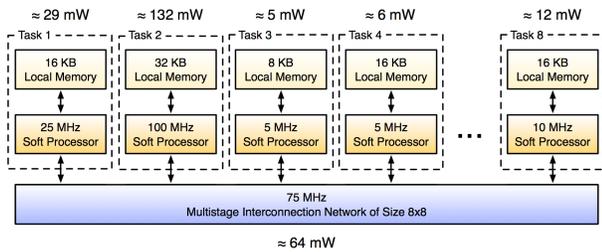


Figure 13. Dynamic power consumption of a MPSoC2 on Virtex-4 FPGA.

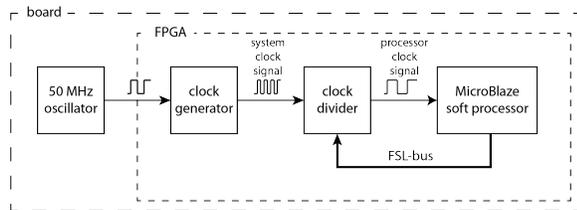


Figure 14. Set-up of the clock rate controller.

$$P_{network} \approx 0.85 \frac{mW}{MHz} \quad (8)$$

V. MPSOC2 WITH SELECTIVE AND ADAPTIVE CLOCKING AND SOFTWARE-FIRST DESIGN

Space-sharing allows selective and adaptive clocking of every PMM by means of a clock rate controller for every PMM. Furthermore, the software-first design method prescribes to configure each PMM such that it matches exactly the requirements of the task it has to execute. To test the influence of both methods, we defined MPSoC2, which has eight PMMs that execute eight different example tasks in their memories. Memory sizes and clock frequencies were chosen as needed by the tasks. In Fig. 13, the resulting MPSoC2 is depicted.

A. Clock Rate Controller

Fig. 14 shows the block diagram of the clock rate controllers of MPSoC2. Every controller derives its input from a system clock generator, which is global for the FPGA and generates its output by a clock divider. The divider can be set either to a constant rate or, as depicted in Fig. 14, can be controlled by the application software during program execution. In second case the clock divider is coupled to the soft processor via Xilinx FSL-bus. Thereby the clock rate can be set just by sending the new value, which is done within two clock cycles at least.

B. Task Segmentation

To benefit from individual clocking, the application programmer must set the clock frequency for every task explicitly. If the task has several phases with different time constraints then he can set clock rates that are adapted to each phase. To accomplish this, it is required to segment the task into time intervals in which the same rate holds as it is shown in Fig. 15. After that task segmentation, the programmer can define the clock rates by two ways: either he uses a procedure call, which is executed during runtime, or he uses a compiler directive, which is evaluated during compile time. Both methods augment the original code. Other examples of code augmentation by time constraints that can be found in literature are given in [13].

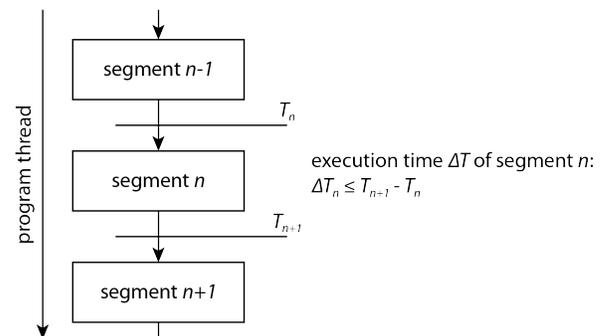


Figure 15. Clock rate adaptation by task segmentation.

Fig. 16 shows an example code where a library procedure `set_clock_rate` is responsible for clock rate setting. The application itself `check_new_data` periodically checks a sensor for the arrival of new data. This may be accomplished at a slow sampling rate of 2 MHz, for example. If new data are present, the application `calc_value` is executed at a higher rate of 40 MHz, for example, in order to process the incoming data quickly. By this method, the processor clock varies dynamically during program execution. Thus the dynamic power dissipation rises and falls with the computing requirements.

The disadvantage of this method is that the programmer must identify clock change points in his code and must specify their value manually. Moreover, the clock rates depend on the used PMM because a superscalar processor, which is more powerful would execute more instructions in the same time. This means that clock rate settings have to take into account the concrete PMM on which the task is executed, what makes applications difficult to port between PMMs.

C. Compiler directive for Clock Rate Adaptation

A better method for adaptive clocking is adding a time ruler to the code according to Fig. 15. Each tick in the time ruler T_i defines a point in time until a program phase

```

set_clock_rate(2);
while(new_data = 0){
  new_data = check_new_data(&data);
};
set_clock_rate(40);
value = calc_value(&data)
set_clock_rate(2);

```

=> clock rate = 2 MHz

=> clock rate = 40 MHz

=> clock rate = 2 MHz

$$P_{dyn} \approx 1909mW \quad (9)$$

comprising of

$$P_{processor} \approx 800mW \quad (10)$$

$$P_{memory} \approx 1024mW \quad (11)$$

$$P_{network} \approx 85mW \quad (12)$$

Figure 16. System call for setting processor clock rate.

must be completed. The adding of time values as ticks is accomplished manually by a compiler directive. The directives are formatted as a comment with a following \$ symbol, for example, to avoid confusion with language extensions or with system calls but they are not treated as comments or extensions. Instead, only a compiler pre-processor reads all meaningful comments and evaluates them. Furthermore, the compiler knows best the PMM type it has to generate code for. This allows for an automatic calculation of the desired clock rate for every program phase. The calculation is performed by the pre-processor in accordance to the respective execution speed of the PMM. The pre-processor counts the number of clock cycles needed to execute every phase due to the time ruler. With that information, the pre-processor can then set the clock rate for every phase, and the programmer does not have to care about the concrete PMMs computation speed, as long as it is fast enough.

```

/*$ initiate */
while(new_data = 0){
  new_data = check_new_data(&data);
};
/*$ event ≤ initiate + 20 */
value = calc_value(&data)
/*$ terminate ≤ event + 30 */

```

=> 40 clock cycles / 20μs = 2 MHz

=> 1200 instructions / 30μs = 40 MHz

Figure 17. Compiler directive for setting processor clock rate.

Fig. 17 shows the example of Fig. 16 with the additional time ruler.

VI. COMPARISON OF MPSOC1 WITH MPSOC2 UNDER THE SAME USER APPLICATION

In the following, we compare the dynamic power consumption of MPSoC1 with that of MPSoC2 in order to explore the effect of adaptive and selective clocking, together with the effect of the software-first design methodology. For a fair comparison, MPSoC1 and MPSoC2 got the same eight tasks to execute, and we measured the total dynamic power dissipation during their execution. MPSoC1 is based on a Virtex-4 FPGA with a fixed clock rate of 100 MHz for all system components and a static memory size of 128 KB for all eight PMMs. MPSoC2 is based on the same FPGA but dependent of its task requirements with clock rates of 25, 100, 5, 5 and 10 MHz and with memory sizes of 16, 32, 8, 16, 16 KB. Our measurements at MPSoC1 showed that the total dynamic power consumption is:

This means that on average every PMM has a dynamic power dissipation of 228 mW. In contrast to that is the example of MPSoC2 (cf. Fig. 13) that produced the same results within the same time limits as MPSoC1. However, it consumed only a total of 184 mW, resulting in an average of 23 mW per PMM. Many other examples can be found that show the same trend. According to that it can be stated that adaptive and selective clocking together with software-first design are efficient methods to reduce the system's energy consumption in space-sharing.

VII. CONCLUSION AND OUTLOOK TO FUTURE WORK

In this paper, the methods of space-sharing, adaptive and selective clocking and software-first design were proposed and demonstrated in two example multiprocessor systems that resided on a single FPGA. The combination of these methods into a methodology allows to get at the same time high computing power and low electric power dissipation. This new result is valid for the class of embedded real-time systems because smaller tasks have to be executed there. Nevertheless, complex feed forward and feed back control systems can be established that way, comprising hundreds of tasks with interprocess communication. All tasks are executed in parallel on by processor-memory modules (PMMs) that are connected via a special multistage interconnection network that is real-time capable. Interprocess communication is either possible by passing messages through the network or by shared variables that reside in a global (external) memory.

Space-sharing means that exactly so many PMMs are synthesized on one FPGA as there are user tasks, including all interrupt service routines and device drivers that are needed by the user application. Adaptive and selective clocking means that every PMM is clocked individually and with a rate that matches the needs of the task it executes, even if these needs vary from task phase to task phase. Software-first design means that the PMM is configured in its architecture and in its memory size such that it meets exactly the task requirements, thus avoiding waste of chip and energy resources.

As a result, space-sharing eliminates the need to find a proper schedule for a set of tasks that must be executed until a given time interval or at a given time point. It

also eliminates the analysis of worst-case execution time (WCET) in embedded controllers, which can be very complex. Additionally, space-sharing allows for a better electrical power management and for memory protection because of the spatial isolation of tasks. Finally, dynamic power dissipation can be reduced in space-sharing by means of adaptive and selective clocking and by a software-first design to the absolute minimum. This is applicable, e.g., in car electronics, where an increasing amount of real-time tasks has to be proceed at a minimum of energy use.

Future work will create a tool set that automatically synthesizes a FPGA-based MPSoC from a XML description of tasks and from a second description of the tasks' memory and time constraints. The resulting embedded controller will be easily synthesizable because it consists only of small and individual processor-memory-modules. It will though exhibit high computing power and low energy dissipation at the same time.

REFERENCES

- [1] S. Aust and H. Richter, *Space Division of Processing Power For Feed Forward and Feed Back Control in Complex Production and Packaging Machinery*, Proc. World Automation Congress (WAC 2010), Kobe, Japan, Sept. 2010, pp. 1-6.
- [2] S. Aust and H. Richter, *Energy-Aware MPSoC with Space-Sharing for Real-Time Applications*, The 5th International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2011), Lisbon, Portugal, Nov. 2011, pp. 54-59.
- [3] P. Marwedel, *Embedded System Design*, 2nd edition, Dordrecht; Heidelberg: Springer, 2011.
- [4] G. C. Buttazzo, *Hard Real-Time Computing Systems. Predictable Scheduling, Algorithms and Applications*, Boston; Dordrecht; London: Kluwer Academic Publishers, 1997.
- [5] Rapita Systems Ltd., www.rapitasystems.com (last checked: 11-06-20).
- [6] Symtavision GmbH, www.symtavision.com (last checked: 11-06-20).
- [7] Xilinx Inc., *MicroBlaze Processor Reference Guide*, October 2009.
- [8] H. Richter, *MULTITOP - Ein Multiprozessor mit dynamisch variabler Topologie*, Dissertation, Fakultät fuer Elektrotechnik und Informationstechnik der TU Muenchen, 1988, (in German).
- [9] H. Richter, *MULTITOP - A multiprocessor with dynamic variable topology (English Summary)*, IPP Technical Report R/35, Max-Planck-Institut fuer Plasmaphysik, 1988.
- [10] H. Richter, *Interconnecting Network*, US-Patent Nr. 5,175,539, 1992.
- [11] S. Aust and H. Richter, *Real-time Processor Interconnection Network for FPGA-based Multiprocessor System-on-Chip (MPSoC)*, The 4th International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2010), Florence, Italy, Oct. 2010, pp. 47-52.
- [12] S. Aust and H. Richter, *Skalierbare Rechensysteme fuer Echtzeitanwendungen*, in: W. A. Halang (editor): *Herausforderungen durch Echtzeitbetrieb*, Springer, 2011, pp. 111-120, (in German).
- [13] A. Leung, K. V. Palem, and A. Pnueli, *TimeC: A Time Constraint Language for ILP Processor Compilation*, *Constraints*, vol. 7, no. 2, 2002, pp. 75-115, doi: 10.1023/a:1015131814255.
- [14] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, K. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, *Leakage Current: Moore's Law Meets Static Power*, *IEEE Computer*, vol. 36, issue 12, Dec 2003, pp. 68-75.
- [15] L. Shang, A. S. Kaviani, and K. Bathala, *Dynamic Power Consumption in VirtexTM-II FPGA Family*, Proc. of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays (FPGA '02), Monterey, CA, Feb. 2002, pp. 157-164.
- [16] H. Yasuura, T. Ishihara, and M. Muroyama, *Energy Management Techniques for SoC Design*, *Essential Issues in SoC Design*, Springer, 2006, pp. 177-223.
- [17] Xilinx Inc., *XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices*, Xilinx document no.UG627, December 2010.
- [18] A. Coxon, *FPGAs auf Low Power trimmen*, *elektronik industrie*, Huethig, issue 1/2, 2009, (in German).

Footprint-Based Generalization of 3D Building Groups at Medium Level of Detail for Multi-Scale Urban Visualization

Shuang He, Guillaume Moreau, Jean-Yves Martin
L'UNAM Université, Ecole Centrale Nantes, CERMA
Nantes, France

Email: Shuang.He; Guillaume.Moreau; Jean-Yves.Martin@ec-nantes.fr

Abstract—In order to enable multi-scale urban visualization, multiple model representations at different levels of detail (LoDs) need to be produced (like by generalization) in advance or on the fly. At local scale, building groups are involved and at least medium LoD is needed in terms of visual perception. Motivated by such demands, this article proposes a novel method for generalizing 3D building groups at medium LoD (the idea was firstly presented in the work of He et al. [1]). The goal is to reduce both geometric complexity and information density. The emphasis is placed on converting 3D generalization tasks into 2D issues via buildings' footprints. The challenge is how to do the mapping from 3D to 2D without losing the information for going back to 3D, especially for a non-prismatic model at medium LoD.

Instead of treating such model as a whole, two preprocessing steps (model partition and unit division) are introduced to decompose a model into suitable structures for footprint-based generalization. As a result, basic generalizations units are obtained, and each of them is divided into *Top* + *Body*. The *Body* part must be a prism for footprint projection. The *Top* part can include roofs and upper walls, and it can be transplanted onto the extruded model by displacement or be generalized with adjacent *Top* parts. Two common types of building groups are studied and different algorithms are developed for their generalization. Experimental results validate the effectiveness of our approach.

Keywords-3D generalization; building group; footprint; level of detail; multi-scale urban visualization

I. INTRODUCTION

3D city visualization requires different representations of building models at different levels of detail (LoDs) to satisfy different scales and application needs. These LoDs should be generated automatically by specific generalization procedures. Generalization has a long history in cartography [2], with the goal of emphasizing the most important map elements while still representing the world in the most faithful and recognizable way. 3D building generalization in city visualization shares the same goal, but should consider both geographical and 3-dimensional information.

As discussed and listed in [3], unlike 2D maps that have standard official scale series, there are no generally agreed LoDs for 3D buildings. Including the four LoDs defined by CityGML (City Geography Markup Language) [4], the existing definitions of LoDs for 3D buildings only differentiate by 3D details. That is to say, they hardly

respond to geographical generalization, like the generalization regarding a group of 3D building, where topological relations should also be considered. This seems to lead more attention to single building generalization.

A number of algorithms have been developed for 3D building generalization. Most of those algorithms deal with single buildings. Generalization of building groups is seldom addressed. In 3D city visualization, the goal of generalization is not only to simplify individual objects, but also so to achieve better cognition by emphasizing important features. Thus, there rises a generalization need for building groups. Both 3-dimensional detail and geographical relations should be taken into account. More generalization operations like selection, aggregation, typification and their combinations are expected.

Footprint has been serving as the connection between 2D and 3D. Plenty of block models of buildings were extruded from cadastral maps using their footprints and heights. But more detailed models could not be acquired in this way. Therefore, a question rises here: how can we translate 3D building generalization issues into 2D scope for generalizing more detailed 3D building models?

This article is organized as below: related work is reviewed in Section II. Concerned issues are discussed in Section III. Section IV gives an overview of our approach. Section V and Section VI focus on decomposing a 3D building model into suitable units for footprint-based generalization through model partition and unit division. Generalization algorithms for two types of building groups are developed in Section VII. Section VIII implements the proposed approach and presents the results. Section IX concludes the article. Section X discusses the future work.

II. RELATED WORK

Compared with the history and achievements in 2D generalization, 3D generalization is still very young and immature. A number of algorithms have been developed for generalizing single buildings. Thiemann proposed to segment a building into basic 3D primitives and to decompose the whole generalization process into segmentation, interpretation and generalization phases [5]. Mayer [6] and

Forberg [7] developed scale-space techniques for simplifying buildings, partly based on the opening and closing morphological operators. Kada proposed to define parts of simplified buildings as intersections of half-planes [8] and to divide buildings into cells and to detect features by primitive instancing [9]. Without semantic information, these methods mainly detect building features based on pure geometry.

By taking semantic information into account, Fan et al. [10] proposed a method for generalization of 3D buildings modeled by CityGML from LoD3 to lower LoDs. Their research showed that good visualization properties could be obtained by only using the exterior shell of the building model that drastically decreases the required number of polygons. Fan and Meng [11] extended their work to the generalization of CityGML LoD4 building models, and concentrated on deriving LoD2 CityGML buildings from LoD3 [12]. However, the above mentioned methods are all limited to generalization regarding single buildings.

Anders [13] proposed an approach for the aggregation of linearly arranged building groups. Their 2D silhouettes, which are the results of three orthogonal projections, are used to form the generalized 3D model. Guercke et al. [14] studied the aggregation of LoD1 building models in the form of Mixed Integer Programming (MIP) problems.

Techniques start emerging for generalizing 3D building groups in the context of city visualization. Glander and Döllner [15] proposed cell-base generalization by maintaining a hierarchy of landmarks. In each cell, only landmark buildings can be seen, the other buildings are replaced by a cell block. In the work of Mao et al. [16], buildings are divided into clusters by road network, and grouped with close neighbors in each cluster. However, only LoD1 buildings were handled.

Moreover, many other algorithms have been developed emphasizing different aspects. Putting the emphasis on progressively removing details, Sester and Klein [17] introduced a rule base which can guide facade generalization including aggregation of neighboring windows, elimination, enlargement or displacement of small facade features depending on their relative importance. Kada [18] introduced an algorithm of constrained invasive edge reduction. Rau et al. [19] focus on automatic generation of pseudo-continuous LoD polyhedral 3D building models, using only one parameter, i.e. feature resolution. For the purpose of simplifying and emphasizing 3D buildings, Thiemann and Sester [20] presented adaptive 3D templates. They categorize building models into a limited number of classes with characteristic shapes, and then use these templates for typical 3D buildings and replace the original 3D shape with the most similarity of those templates. Zhang et al. [21] studied geometry and texture coupled generalization towards realistic urban visualization. He et al. [22] proposed a new way to produce LoDs for 3D city models at (pseudo) all range of scales, by combining generalization and procedural modeling.

III. CONCERNED ISSUES

A. Levels of detail for buildings

The existing methods for measuring levels of detail of building models mainly use descriptive expressions, such as listed in the survey of Meng and Forberg [3] and defined in CityGML standard [4]. CityGML differentiates five building LoDs. A LoD0 building can be represented by footprint or roof edge polygon. LoD1 is the well-known blocks model comprising prismatic buildings with flat roofs. In contrast, a building at LoD2 has differentiated roof structures and thematically differentiated surfaces. LoD3 denotes architectural models with detailed wall and roof structures, balconies, bays and projections. LoD4 completes a LoD3 model by adding interior structures for 3D objects. In general, we can consider LoD0 and LoD1 as low LoDs, LoD2 as medium LoD, LoD3 and LoD4 as high LoDs.

The five CityGML's LoDs are generally accepted, however, each LoD obviously covers a rather wide range. Thus, many in-between LoDs can hardly be distinguished. Moreover, these LoDs are made for individual buildings. They will face more challenges when a number of building models are involved. We also adopts CityGML's definitions, but trying to extend them for denoting LoDs for building groups and indicating more in-between levels.

B. Generalization scale and complexity

At different scales, 3D building models have different features and the corresponding generalization faces different types of complexity, such as geographic complexity and geometric complexity. Here we use the term - geographic complexity - to refer to the complexity related to geographic distribution, including topological relation, information density, etc. Geometric complexity refers to the complexity of geometric representation of models, such as the number of primitive elements.

At city scale, low LoD building models are mostly involved, so the generalization task mainly deals with geographic complexity. At object scale (e.g. single buildings), high LoD models are often required, so the generalization focus is on geometric complexity. At local scale (e.g. building groups), medium LoD models are usually concerned, thus both geographic complexity and geometric complexity should be considered. Compared with single building generalization, building group generalization is seldom addressed but also quite needed, for example, when we want to reduce computational complexity without losing the recognizability of a building cluster. Therefore, the proposed generalization approach aims at reducing both geometric and geographic complexity of a group of buildings, meanwhile maintaining the general aspect of the group.

For the reduction of geometric complexity, a number of generalization algorithms have been specifically designed for single buildings. Can we make generic approaches that

suit to both (complex) single buildings and building groups? For the reduction of geographic complexity, can we adapt 2D generalization techniques to 3D scope? Both of these two challenges are concerned in the development of our approach.

C. Object nature and model quality

A proper generalization approach is firstly oriented by the object nature. In 3D computer graphics, a great number of algorithms have been developed for simplifying polygonal representations of solids and surfaces for general 3D objects. However, these algorithms can hardly be applied to buildings, because most 3D building models are already low-polygon models. Besides, parallel and orthogonal properties of buildings need to be respected during simplification. Interdependency between building components, adjacent buildings, and other city objects should also be considered. Therefore, determined by the nature of building, generalization algorithms for 3D building models are usually specifically designed.

Semantics plays an important role in building generalization. If we know the semantic meaning (wall, roof, window, door, balcony etc.) of each geometry, it can be properly treated according to its kind. The existing algorithms for building generalization can be grouped into two categories decided by whether semantic information is provided along with the geometric model. For generalizing pure geometry building models, the primary effort is mainly devoted to feature detection and segmentation. Such effort can be exempted if semantics is provided. Moreover, the coherence of geometry and semantics is also influential. Stadler and Kolbe [23] pointed out that the more information is provided by the semantic layer, the less ambiguity remains for geometrical integration. We believe it is also true for generalization task. Figure 1 shows two extreme cases of the level of coherence.

IV. APPROACH OVERVIEW

The generalization target of this approach is a group of building models at medium LoD. The emphasis is placed on translating 3D generalization tasks into 2D problems. The strategy is to generate footprints of 3D buildings, perform 2D generalization on their footprints, and then extend the result to 3D.

Figure 2 depicts the main flow of the proposed footprint-based approach for generalizing building groups. Because the available 3D building models usually come with unfavorable data structures for direct footprint projection, preprocessing is needed to obtain suitable units for projection and generalization (step 1). Because we are coping with building models at medium LoD, roof structures should be properly handled. Another preprocessing method (step 2) is introduced to divide each generalization unit into *Top + Body*, so that footprints can be projected from each *Body* (step 3), and *Top* parts can be preserved for separated

generalization. Any 2D generalization operators and algorithms can then be applied to their footprints (step 4), hence allowing arbitrary levels of generalization. Prismatic bodies can be extruded from generalized footprints (step 5). The *Top + Body* division supports flexible roof generalization (e.g. displacement and flattening) on *Top* parts (step 6). Assembling the results of extrusion and *Top* generalization (step 7), the final result of the generalized 3D building group can be obtained.

V. PREPROCESSING I: MODEL PARTITION

The goal of partitioning is to decompose the original model into basic units favorable for generalization. Here rises the question: what makes a favorable unit for generalization? We believe there are two important criteria for such unit: 1) it has coherent structure in geometry and semantics; 2) it (or a part of it) can be fully represented by its footprint. The first criterion is one of the emphases addressed by CityGML [24]. The second criterion is seldom mentioned, but very important for facilitating generalization tasks and for adapting 2D generalization techniques into 3D scope. Using CityGML LoD2 building model as example, partition rules are developed, so that basic units can be obtained with beneficial attributes for generalization.

A. CityGML LoD2 building model

In order to develop proper strategy for partitioning, we should be aware of the data structure and all possible elements of the target model. The original UML diagram of CityGML building model [4] includes all four LoDs. To be more concentrated, we redraw a UML diagram exclusively for LoD2. See Figure 3.

In Figure 3, «*Geometry*» implies purely geometric representation, whereas «*Feature*» implies geometric/semantic representation. According to CityGML standard [4], the concerned features are explained as below:

A *Building* can consist of many *BuildingParts*, and each *BuildingPart* can also consists of many *BuildingParts*. That is to say, a hierarchical building tree of arbitrary depth may be realized. Moreover, each *Building/BuildingPart* could consist of three types of elements: text attributes, pure geometry, and geometric/semantic features (for LoD2, they are *BuildingInstallations*, *_BoundarySurfaces*, and *BuildingParts*). All these elements will be treated differently in our generalization approach. Figure 4 depicts all possible elements and their treatments in the following generalization.

B. Partition rules

Above all, we would like to clarify that the following rules are designed to partition common buildings. It is reasonable to believe that a common building has all the main walls starting from and orthogonal to the (local) ground

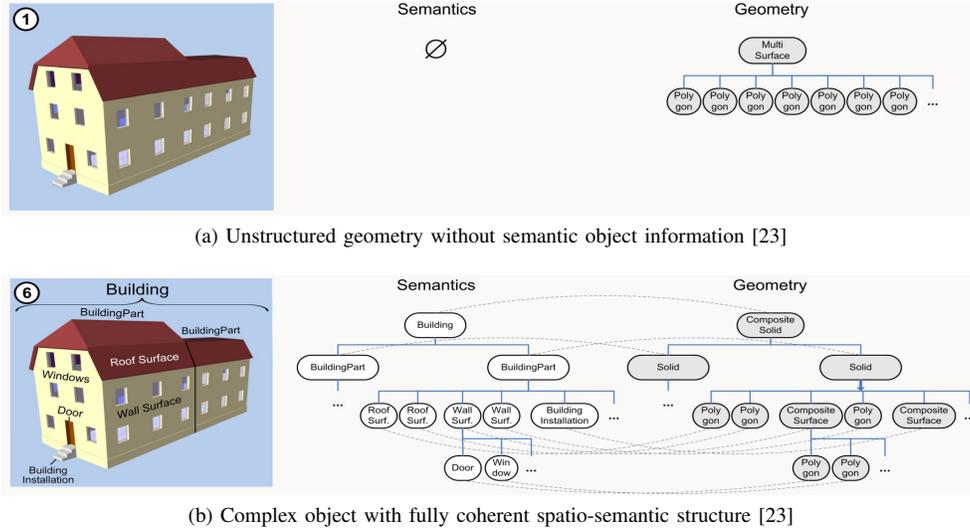


Figure 1: Comparison of two extreme cases in terms of spatio-semantic coherence.

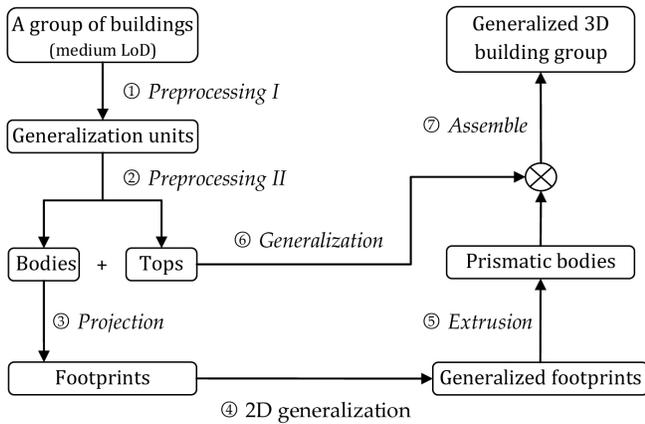


Figure 2: Main flow of the approach for generalizing building groups.

plane. Otherwise, we'd better consider this building as an uncommon building, and prescribe different treatment for generalization. In many cases, uncommon buildings are landmark buildings which may not need to be generalized.

The goal of partition is to get a well structured building in both semantic and geometric sense, so as to extract suitable unit for generalization. As mentioned above, we believe a favorable generalization unit should be able to be fully represented by its footprint. Even though a CityGML building model already has semantically structured geometry, the geometry of each feature may still not be applicable due to the different habits in modeling process. For instance, a complex building may not be decomposed into different building parts; the protruded surfaces of a balcony may be modeled as a part of the wall surfaces. In order to regulate

such kind of geometry, the partition rules are introduced as below:

- If a *Building* is composed of unconnected segments, partition them into different *Buildings*.
- If a *Building* is composed of structural segments differing in e.g. height or roof type, partition them into different *BuildingParts*.
- If a *Building/BuildingPart* has smaller components which are not significant as *BuildingParts* (e.g. chimneys, dormers, and balconies), partition them into *BuildingInstallations*.
- If a *Building/BuildingPart* has geometries without semantic information, partition them into pure geometries.
- If a *Building/BuildingPart* has *BuildingParts* and *BoundarySurfaces* at the same level, make the *BoundarySurfaces* into a new *BuildingPart*.
- If a *Building/BuildingPart* includes only one *BuildingPart*, move the included *BuildingPart* into its parent *Building/BuildingPart*.
- If a *Building* has *BoundarySurfaces*, there must be a *WallSurface* starting from and orthogonal to the ground plane; otherwise, partition this *Building* as a *BuildingInstallation* into another *Building*.
- If a *BuildingPart* has *BoundarySurfaces*, there must be a *WallSurface* starting from and orthogonal to the ground plane; otherwise, partition this *BuildingPart* into a *BuildingInstallation*.
- If a *Building/BuildingPart* has unconnected or self-intersected *WallSurface*, partition it into more *BuildingParts*.

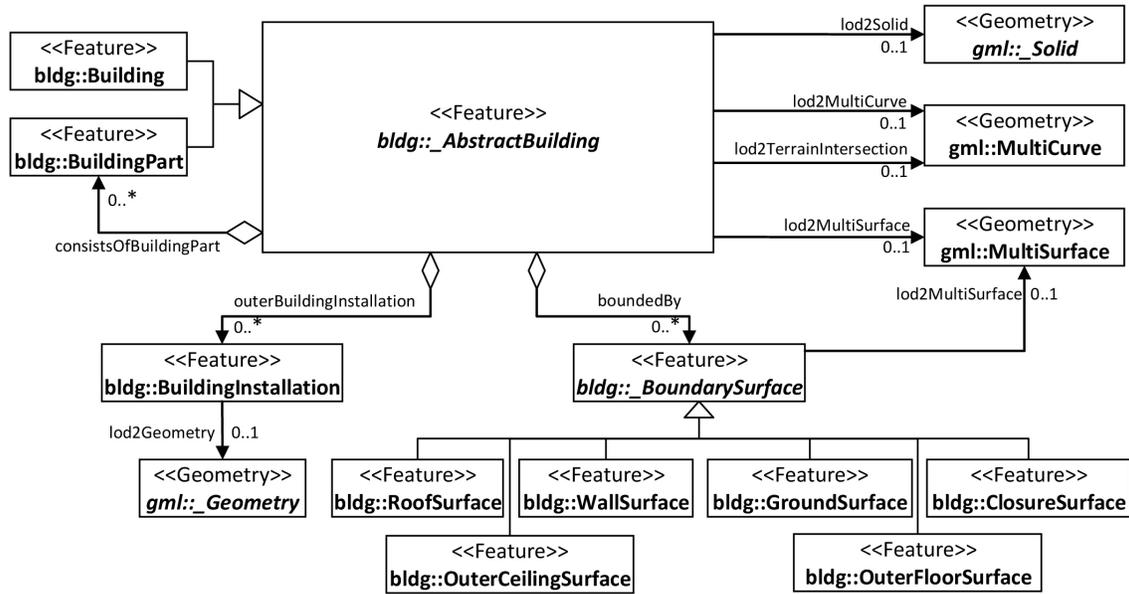


Figure 3: UML diagram of CityGML LoD2 building model drawn based on CityGML standard [4]

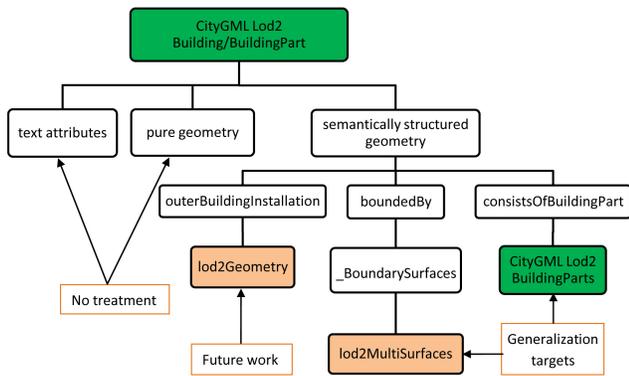


Figure 4: All possible elements of a CityGML LoD2 building model and their treatments in generalization.

C. Partition result

After employing the proposed partition rules, a building tree can be obtained consisting of basic generalization units with beneficial attributes. Although each *Building/BuildingPart* could include text attributes, pure geometry, *BuildingInstallations*, and *_BoundarySurfaces*, only *_BoundarySurfaces* are selected to form the basic generalization unit. The term *unit* will be used in the following discussion, referring to a node of a building tree only consisting of *_BoundarySurfaces*. A simple building only has one generalization unit, which is the root node of its building tree. A complex building has at least two generalization units, including all leaf nodes of

its building tree, as illustrated in Figure 5.

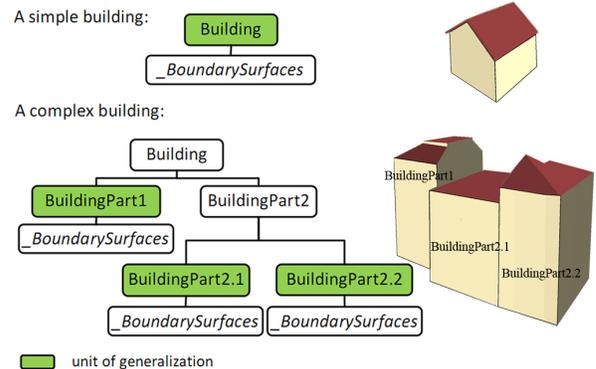


Figure 5: Two examples of building trees consisting of basic generalization units.

The beneficial attributes of a basic generalization unit are listed as below:

- Each basic unit only consists of *_BoundarySurfaces*.
- Each basic unit has unique height.
- If there are *_BoundarySurfaces*, there must be a *WallSurface*; other types of surfaces are optional.
- A *WallSurface* must start from and be orthogonal to the ground plane.
- The orthogonal projection of *WallSurfaces* of each leaf node form a simple polygon or polyline.

So far, we have obtained basic units with favorable *WallSurfaces*, but each unit still cannot be fully rep-

resented by footprint fp , which is obtain by projecting all its walls without considering the roof. There also rises the issue of roof generalization. We propose to use fp to fully represent only a part of the unit and to handle roof generalization separately in preprocessing II.

VI. PREPROCESSING II: UNIT DIVISION

A common way of roof generalization is by primitive matching of different roof types. But type detection is a costly (most often manual) and uncertain process depending on the given types and lots of parameters. In CityGML building models, roof surfaces are separated from walls, but roof type is not always available in attributes. Even if given the roof type, the rebuilding of roof after extrusion would be another difficulty without knowing parameters.

Our approach places the emphasis on converting 3D generalization tasks into 2D scope via footprints. But direct footprint projection of basic generalization units will still lose the roof information, which is important when extending 2D generalization results to 3D. We propose to divide each generalization unit into $Top + Body$ before conducting footprint projection. The $Body$ part is a prismatic model, which can be fully represented by its footprint and associated height. The Top part consists of the rest structures (e.g. roofs and upper walls) of the unit, which can be easily transplanted to a prismatic model and can also be generalized with adjacent Top parts. Thus, only the $Body$ part of each unit is used for footprint projection, and the Top part is preserved for roof generalization.

For a building, if all of its walls end at the top in the same horizontal plane, the Top only consists of its roof; otherwise, the Top consists of its roof and the end walls. An example is given in Figure 6.

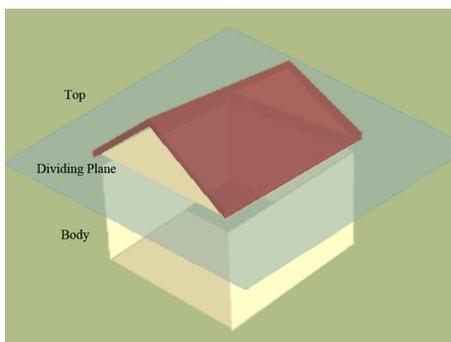


Figure 6: An example of dividing a building into $Top + Body$.

VII. GENERALIZATION ALGORITHMS

After having partitioned each original model into basic generalization units, and having performed $Top + Body$ division on all the units, footprint-based generalization can then be enabled. Generalization tasks always need human

analysis and human decision on what to generalize and how to generalize. Therefore, different strategies and algorithms should be developed for generalizing different types of building groups.

Here we study two types of building groups which widely exist especially in European cities (as illustrated in Figure 7): 1) traditional building groups and 2) modern building groups. It is obvious that type 1) groups have low and crowded buildings with similar heights, but type 2) groups have buildings with prominent difference in height. Therefore, we develop two generalization algorithms for these two types of building groups.

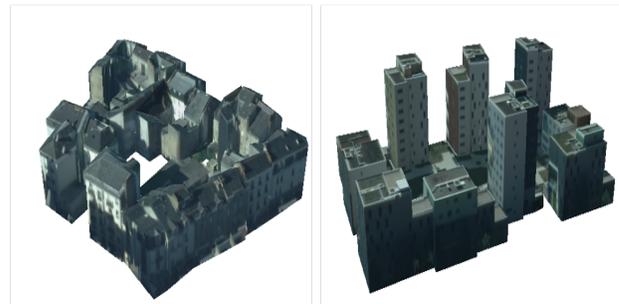


Figure 7: A group of traditional buildings (left) and a group of modern buildings (right), Nantes, France ©IGN BATI 3D

A. Generalization of building groups with a minor difference in height

If a group of buildings have no big difference in height, we believe its outer feature can represent the whole group to a certain extent, like in local scale and or city scale visualization. Therefore, the first generalization task is to detect its outer feature. After have obtained basics generalization units through model partition (Section V), the outer units can be considered as the outer feature of a building group. The outer units can also be aggregated. If all units are outer units, no units will be eliminated, and aggregation can be directly performed on all units. If the outer units have non-flat roofs, two levels of aggregation can be achieved: with or without roof structures. The original roof structures can be preserved based on the $Top + Body$ division (Section VI). They can be transplanted onto the extruded building blocks, and be generalized to flat roofs as well.

Our generalization operations start from a group of LoD2 buildings, but the goal is not to generalize each building to LoD1. Not only each individual building is concerned, but also the overall feature of the group is addressed. We believe that it is better to specify the target when using the concept of LoD, and thereby introduce a more dedicated term - Group LoD - to denote the level of detail of a building group. Group LoD1 and Group LoD2 are defined similar to LoD1 and LoD2 of a building model in CityGML standard.

Another three Group LoDs are introduced to describe more inter-level status. The definitions are given as below:

- Group LoD2: every building model in the group is at LoD2.
- Group LoD1C: the building group is represented by its outer units.
- Group LoD1B: aggregated Group LoD1C model with the same height, but differentiated roof structures.
- Group LoD1A: Group LoD1B model with flattened roof.
- Group LoD1: the building group is a block model, which looks like a LoD1 building model.

The main flow of the algorithm is depicted in Figure 8.

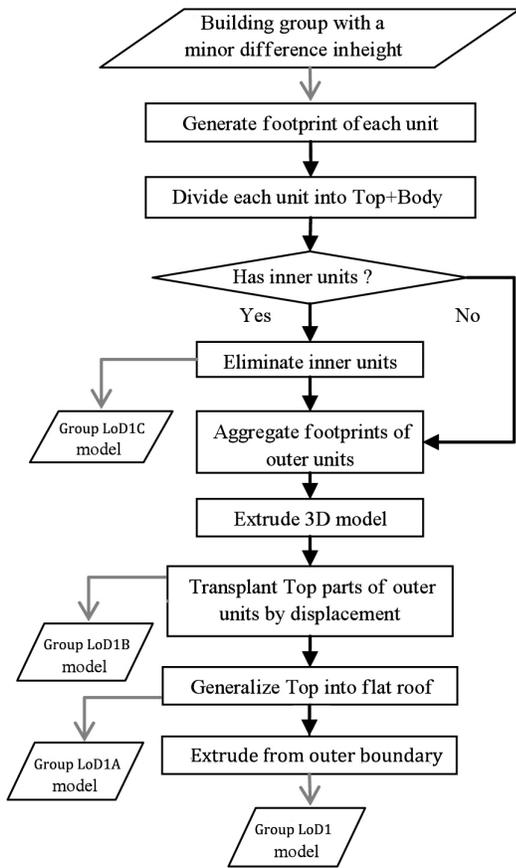


Figure 8: The main flow of the generalization algorithm for building groups with a minor difference in height.

B. Generalization of building groups with a major difference in height

If a building group has a major difference in height, the generalization will be performed on its subgroups. A

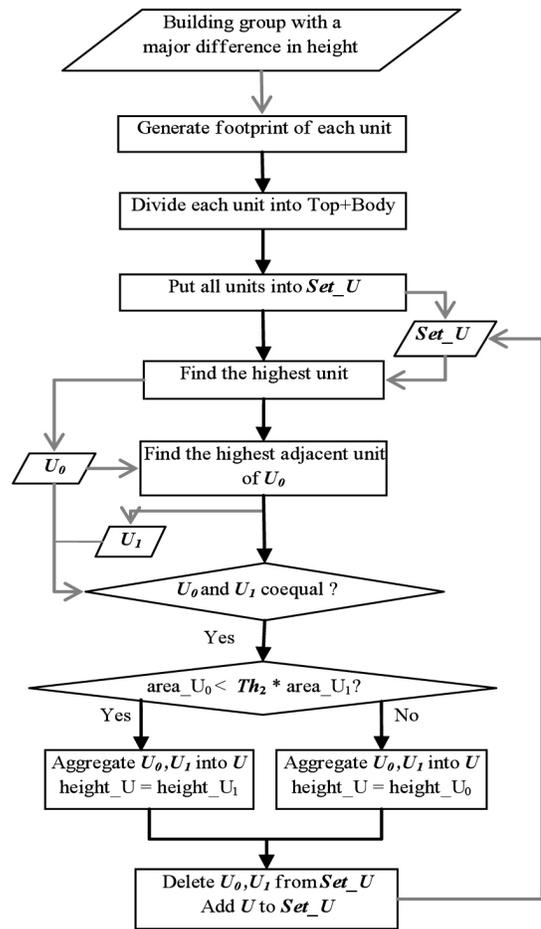


Figure 9: The main flow of the generalization algorithm for building groups with a major difference in height.

subgroup is composed of a center unit and its adjacent neighbors. Each time the highest unit will be chosen from the unprocessed units. If this unit is much higher than its neighbor, they should not be aggregated. We use the term coequal in this paper to indicate that the height difference in two units can be ignored, that is to say, they can be aggregated. Coequal units are defined as below:

Given two units U_1 with height h_1 and U_2 with height h_2 , if they satisfy the constraint as in Equation (1), U_1 and U_2 are coequal, where Th_1 is a predefined variable as the threshold.

$$\frac{1}{Th_1} < \frac{h_1}{h_2} < Th_1 \tag{1}$$

When merging two adjacent and coequal units, either height of the original units can be assigned to the new unit. If the lower unit covers a rather large area, the new unit

takes the lower height; otherwise, it takes the higher one. We propose a criterion as below:

Given a_1 , a_2 , and h_1 , h_2 ($h_1 < h_2$) as the areas and heights of two coequal units, the height of new merged unit h_3 is determined as in Equation (2), where Th_2 is a predefined variable as the threshold.

$$h_3 = \begin{cases} h_1 & \text{if } a_1 \geq Th_2 * a_2 \\ h_2 & \text{otherwise} \end{cases} \quad (2)$$

The main flow of the algorithm is depicted in Figure 9.

VIII. EXPERIMENTAL RESULTS

The proposed footprint-based generalization approach is tested on two sets of building groups and on a land parcel consisting of 290 buildings.

A. Test 1: Generalization of Building Groups with a Minor Difference in Height

The algorithm for generalizing building groups with minor difference in height (Section VII-A) is tested on a building group consisting of 381 generalization units. The statistics are given in Table I, and the results are shown in Figure 10.

B. Test 2: Generalization of Building Groups with a Major Difference in Height

The algorithm for generalizing building groups with major difference in height (Section VII-B) is tested on a building group consisting of 193 units. The results are shown in Figure 11, and the statistics are given in Table II. In this test we define $Th_1=2$, and $Th_2=4$. Please see Equation (1) and (2) for the meaning of Th_1 and Th_2 .

C. Test 3: Generalization of Building Groups in A 3D City

A parcel of land consisting of CityGML LoD2 buildings is integrated into a 3D city model, and generalization can be performed on these LoD2 buildings (Figure 12b). With the purpose of conveying the overall feature of building groups, three inter-level models between Group LoD2 and Group LoD1 are generalized using the algorithm proposed in Section VII-A. The results are shown in Figure 12. Table III presents some statistics for each Group LoD, including the number of polygons, buildings, and generalization units. All adjacent buildings are merged into new buildings at Group LoD1C. The following generalization operations are performed on each of these nonadjacent buildings, so the number of building does not change then. But the number of generalization unit drops at each Group LoD, which implies the reduction of information density. The drop of polygon number indicates the reduction of computational complexity.

IX. CONCLUSION

This article presented a novel approach for generalizing 3D building groups. The goal is to reduce both geometric complexity and information density, meanwhile maintaining recognizability, which requires at least medium LoD models. The emphasis has been placed on translating 3D generalization issues into 2D scope via footprints. First of all, a meaningful partition was suggested so that each footprint can carry feature information. A set of partition rules was developed for partitioning the buildings modeled by CityGML at LoD2.

Footprint-based generalization is then confronted with the difficulty of roof generalization. Unlike the existing approaches such as primitive matching, we proposed to divide a building into *Top+Body*. Thus, a *Top* part can be easily transplanted onto the extruded model by displacement. Of course, a *Top* part can also be generalized to a flat roof, or be aggregated with adjacent *Top* parts.

Two types of building groups were distinguished and further studied in this work: one has major difference in height and the other has minor difference in height. For the former one, we believe its outer feature can represent the whole group to a certain extent. For the latter one, it should not be handled as a whole. An iterative aggregation process is performed by comparing the height and area of every two adjacent units starting from the highest one.

The approach was tested on two building groups and a part of 3D city model. Group generalization shows its advantage in reducing information density, e.g. by eliminating insignificant buildings. Different from the methods only handle LoD1 block models, our approach can handle LoD2 models as well. Instead of aggregating detailed models directly into LoD1 blocks, our approach supports generalization of Group LoDs with geographical features, thereby achieving data reduction and maintaining recognizability at the same time.

As mentioned in Section III-B, we aimed at finding solutions to these two challenges: 1) Can we make generic approaches that suit to both (complex) single buildings and building groups? 2) Can we adapt 2D generalization techniques to 3D scope? Towards the first challenge, we proposed model partition to decompose a building model into generalization units. Thus, a complex building can be seen as a connected building group. Towards the second challenge, we divided each unit into *Top+Body*, so that the *Body* part can be fully represented by its footprint, through which 3D generalization tasks can be converted to 2D issues.

X. FUTURE WORK

During unit division, very small upper walls could be generated mostly due to the irregularity of the input model. Those segments are expected to be discarded, but additional roof adjustments are required to keep the topological relation between walls and roof. This will be our next work.

The proposed methods of model partition and unit division are developed for common buildings, with walls starting from and orthogonal to the (local) ground plane. Buildings with special architectural designs will be studied in the future, as well as the necessity of their generalization in the context of urban visualization, since they are usually landmark buildings.

For building group generalization, the ratio of area between space and inner units will be considered during generalizing building groups with a minor difference in height; only two types of building groups have been studied, so more types should be studied in the future; only connected buildings are treated, and building groups with disjoint buildings will be concerned.

Towards LoD measurement of 3D building models, this work introduced a more dedicated term Group LoD and several new Group LoDs to describe in-between levels. There remains open questions: e.g. how to measure LoD with more precision? How many LoDs are there? How to decide LoD according to scale?

REFERENCES

- [1] S. He, G. Moreau, and J.-Y. Martin, "Footprint-based 3d generalization of building groups for virtual city visualization," in *GEOProcessing 2012*, 2012, pp. 177–182.
- [2] R. B. McMaster and K. S. Shea, *Generalization in Digital Cartography*. Washington, D.C.: Assoc. of American Geographers, 1992.
- [3] L. Meng and A. Forberg, *3D Building Generalisation*. Elsevier, 2007, ch. 11, pp. 211–232.
- [4] *OSG City Geography Markup Language (CityGML) Encoding Standard (Version 2.0.0)*, Open Geospatial Consortium Inc. Std. OGC 12-019, 2012.
- [5] F. Thiemann and M. Sester, "Segmentation of buildings for 3d-generalisation," in *Proceedings of 7th ICA Workshop on Generalisation and Multiple Representation*, 2004.
- [6] H. Mayer, "Scale-spaces for generalization of 3d buildings," *International Journal of Geographical Information Science*, vol. 19, no. 8-9, pp. 975–997, 2005.
- [7] A. Forberg, "Generalization of 3d building data based on a scale-space approach," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2004.
- [8] M. Kada and F. Luo, "Generalization of building ground plans using half-spaces," in *Proceedings of the International Symposium on Geospatial Databases for Sustainable Development*, 2006.
- [9] M. Kada, "Scale-dependent simplification of 3d building models based on cell decomposition and primitive instancing," in *Proceedings of the International Conference on Spatial Information Theory*, 2007.
- [10] H. Fan, L. Meng, and M. Jahnke, *Generalization of 3D Buildings Modelled by CityGML*. Springer, 2009, pp. 387–405.
- [11] H. Fan and L. Meng, "Automatic derivation of different levels of detail for 3d buildings modeled by citygml," in *Proceedings of 24th International Cartographic Conference*, 2009.
- [12] H. Fan and L. Meng, "A three-step approach of simplifying 3d buildings modeled by citygml," *International Journal of Geographical Information Science*, vol. 26, no. 6, pp. 1091–1107, 2012.
- [13] K.-H. Anders, "Level of detail generation of 3d building groups by aggregation and typification," in *International Cartographic Conference*, 2005.
- [14] R. Guercke, T. Götzelmann, C. Brenner, and M. Sester, "Aggregation of lod 1 building models as an optimization problem," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 2, pp. 209–222, 2011.
- [15] T. Glander and J. Döllner, "Abstract representations for interactive visualization of virtual 3d city models," *Computers, Environment and Urban Systems*, vol. 33, pp. 375–387, 2009.
- [16] B. Mao, Y. Ban, and L. Harrie, "A multiple representation data structure for dynamic visualisation of generalised 3d city models," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 2, pp. 198–208, 2011.
- [17] M. Sester and A. Klein, "Rule based generalization of buildings for 3d-visualization," in *Proceedings of the International Cartographic Conference*, 1999.
- [18] M. Kada, "Automatic generalization of 3d building models," in *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2002.
- [19] J.-Y. Rau, L.-C. Chen, F. Tsai, K.-H. Hsiao, and W.-C. Hsu, "Lod generation for 3d polyhedral building model," in *Advances in Image and Video Technology*, ser. Lecture Notes in Computer Science, L.-W. Chang and W.-N. Lie, Eds. Springer Berlin / Heidelberg, 2006, vol. 4319, pp. 44–53.
- [20] F. Thiemann and M. Sester, "3d-symbolization using adaptive templates," in *Proceedings of ISPRS Technical Commission II Symposium*, 2006, pp. 49–54.
- [21] M. Zhang, L. Zhang, P. T. Mathiopoulos, W. Xie, Y. Ding, and H. Wang, "A geometry and texture coupled flexible generalization of urban building models," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 70, pp. 1–14, 2012.
- [22] S. He, G. Besuievsky, V. Tourre, G. Patow, and G. Moreau, "All range and heterogeneous multi-scale 3d city models," in *Proceedings of Usage, Usability, and Utility of 3D City Models.*, 2012.
- [23] A. Stadler and T. H. Kolbe, "Spatio-semantic coherence in the integration of 3d city models," in *Proceedings of the 5th International Symposium on Spatial Data Quality*, 2007.
- [24] G. Gröger and L. Plümer, "Citygml – interoperable semantic 3d city models," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 71, pp. 12–33, 2012.

Table I: Statistics for Test 1.

Group LoD	Fig.	Footprint		3D model	
		Vertex	Polygon	Vertex	Polygon
Group LoD2	10a	361	44	1565 (100%)	381 (100%)
Group LoD1C	10d	186	18	843 (53.9%)	203 (53.3%)
Group LoD1B	10f	121	3	679 (43.3%)	153 (40.2%)
Group LoD1A	10g	121	3	590 (37.7%)	121 (31.8%)
Group LoD1	10h	70	1	345 (22.0%)	70 (18.4%)

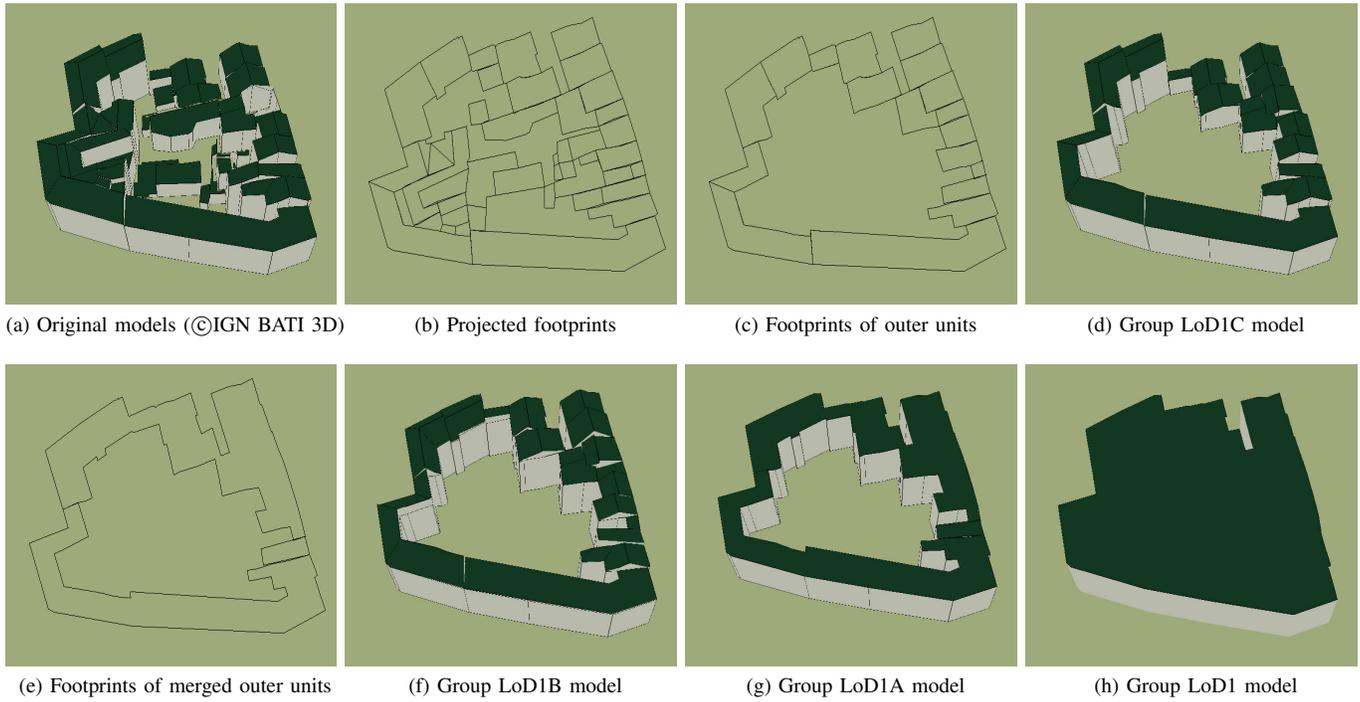


Figure 10: Generalization results of Test 1.

Table II: Statistics for Test 2.

Group LoD	Fig.	Footprint		3D model	
		Vertex	Polygon	Vertex	Polygon
Group LoD2	11a	193	19	879 (100%)	211 (100%)
Generalized	11d	118	5	565 (64.3%)	116 (55%)

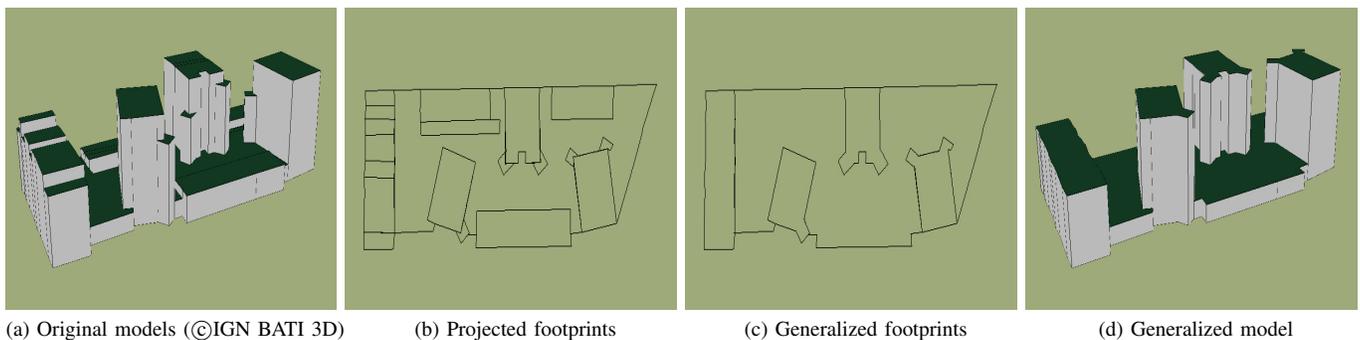


Figure 11: Generalization results of Test 2.

Table III: Statistics for Test 3.

Figure	Group LoD2	Group LoD1C	Group LoD1B	Group LoD1A	Group LoD1
Figure	12b	12c	12d	12e	12f
Polygons	52247 (100%)	37348 (71%)	35592 (68%)	28023 (54%)	20338 (39%)
Buildings	290	213	213	213	213
Units	1421	949	321	312	213

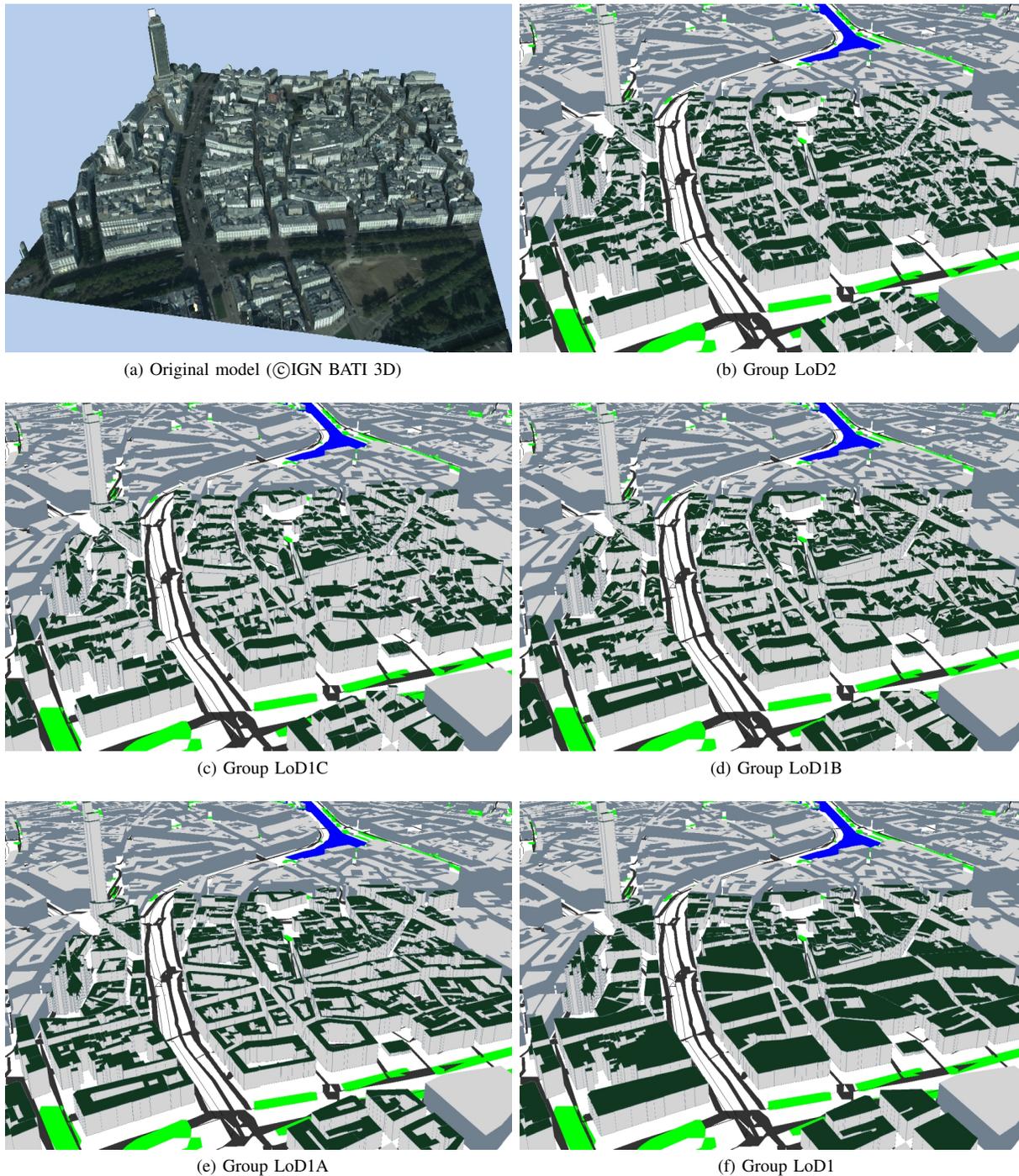


Figure 12: Generalization results of Test 3. Three landmark buildings are marked and excluded from each generalization.

Subjective Assessment of Data Quality considering their Interdependencies and Relevance according to the Type of Information Systems

María del Pilar Angeles, Francisco Javier García-Ugalde
 Facultad de Ingeniería
 Universidad Nacional Autónoma de México
 México, D.F.
pilarang@unam.mx, fgarciau@unam.mx

Abstract — The assessment of Data Quality varies according to the information systems, quality properties, quality priorities, and user experience among other factors. This paper presents a number of user stereotypes, a study of the relevance of the quality properties from experienced users mainly at the industry and an assessment method for subjective quality criteria considering their interdependencies. A Data Quality Manager prototype has been extended to suggest quality properties, their corresponding priorities and the possibility of subjective assessment of secondary quality criteria. The relevance and assessment of quality criteria according to the type of Information Systems is presented and validated.

Keywords-data quality; subjective assessment; user stereotypes; quality framework; data cleansing.

I. INTRODUCTION

The prime motivation for the research is that when users query a database system, they get returned a set of data which is inherently presented as perfect, original, and atomic. Users have no information by which to judge its quality and whether data comes from a number of data sources or by a transformation function. We have developed a Data Quality Manager (DQM) [1], [2], [3], [4] in order to objectively assess data quality within heterogeneous databases. The DQM is composed of a generic Data Quality Reference Model, a Measurement Model, and an Assessment Model. The data quality criteria classification as primary and secondary dimensions has been also previously identified in [5]. The assessment of data quality has been classified as objective and subjective assessment in [11].

The DQM was originally designed to assess primary data quality properties such as currency, response time, volatility. The assessment has been done at different levels of granularity by considering data provenance and aggregation functions. Therefore, the DQM was unable to assess subjective data quality properties.

Further work has shown that the overall assessment of data quality depends on the quality properties chosen as quality indicators, and the priority of each quality property might change the final quality score [2], [5].

Subjective assessment of data quality is not an easy task for naive users without enough experience. Data consumers of a Decision Support System (DSS) might prefer some data against other because of the reputation that data

producers have as well as the credibility and relevance of data for the task at a hand, or the level of satisfaction they have on making strategic decisions effectively from using reliable data. Furthermore, data consumers of operational systems might be more interested in timeliness, response time, and accessibility of data for an effective On-Line Transaction Processing (OLTP) than completeness or relevance of data.

Within the DQM the specification of which quality properties and the priority of those quality properties were meant to be established by expert users. However, in the case of non-expert users, the DQM has no suggestions to make.

In order to establish a data quality assessment tool that can help naive users according to the type information system and to implement the assessment of subjective quality properties to provide more information to expert users, we have established the main objectives of the present paper.

- a) *Data Quality Reference Model improvement*: To identify a set of relevant quality properties according to the type of Information Systems (IS) and the role of user, named as user stereotypes, part of this work has been published in [6].
- b) *Data Quality Assessment Model improvement*: To distinguish a set of data quality priorities from user prototypes, to establish their corresponding weights during the assessment process, part of this work has been published in [1], and to determine a subjective assessment method in order to incorporate secondary quality properties.
- c) *Data Quality Manager Prototype enhancement*: To be able to suggest a set of ranked data quality properties to inexperienced users to assist them with the analysis of a number of data sources to query the best ad-hoc ranked data sources to support informed decision making.
- d) *The implementation of a generic and flexible questionnaire* for the subjective assessment of secondary data quality criteria by the aggregation of its components.
- e) *The implementation of the data quality interdependencies* identified in [6] as part of the subjective assessment method within the DQM.

The following Section is focused on previous research

concerning quality criteria classifications, and types of assessment of quality criteria. The third Section is related to the assessment of quality by considering quality properties interdependencies. The fourth Section presents a questionnaire for the subjective assessment of quality properties considering their interdependencies. The fifth Section presents the ranking of quality priorities according to the IS and role of user. The sixth Section presents a survey that was conducted to provide a ranking of quality properties according to the type of Information System from experienced users. The seventh Section is aimed to the implementation of the relevant quality properties, their corresponding ranking and the subjective assessment within the Data Quality Manager, in order to provide automatic, semi-automatic and manual assessment of data quality. The last Section concludes with main achievements and future work.

II. PREVIOUS WORK

We will present a number of relevant data quality classifications and types of assessments of quality criteria.

A. Data Quality classifications

There are a number of quality criteria classifications, the difference between concepts and classifications rely mainly on user focus according to the role and experience [1], [2]. However, our proposal is focused on the implementation of assessment methods; this Section only presents the assessment oriented model explained in [22] and the data quality classification according to their measure interdependencies.

The Assessment Oriented Model: Quality criteria have been classified in an assessment-oriented model by F. Naumann in [22], where for each criterion an assessment method is identified.

The scores of objective criteria are determined by a careful analysis of data. In this classification the user, the data, and the query process are considered as sources of information quality by themselves.

Individual users determine the scores of subjective criteria based on their experience, knowledge, and focus. Therefore, subject assessment is recommended in case of experienced users.

Object-criteria and process-criteria have been utilized for an unbiased assessment of data within the DQM for any level of user experience. However, subjective criteria had not been considered within the original DQM.

The measurement dependencies classification: The measurement of a quality criterion might be part of the measurement of an aggregate one. The quality dimensions, which measurements derive from primary criteria, are identified as secondary quality properties [2], [5]. We have identified some relationships between these quality properties based on their definitions from previous research [7] [8], [9], [10], [11], [12], [13] [14], where the quality

properties are only defined. However, there is no identification of interdependencies. The metrics or assessment methods identified in previous research were established with no consideration of interdependencies among subjective quality criteria. The secondary quality criteria definitions and their relationship with primary criteria are as follows:

Primary Quality Criteria: From the Data Quality Reference Model, we have identified a number of criteria, which measurement does not depend on other quality criteria, namely Primary Quality [2], some of them are presented in Table 1.

Table 1 PRIMARY QUALITY CRITERIA

Accuracy	Format Precision
Currency	Format Flexibility
Efficient use of storage	Volatility
Response time	Representational Consistency
Availability	Concise Representation
Amount of data	Appropriateness of Format
Unbiased data	Uniqueness

Secondary Quality Criteria: This Section presents a set of secondary quality criteria, their conception and measurement are established on a primary or secondary quality property. These properties are mainly assessed by subjective methods and some of them are presented in Table 2.

Table 2 SECONDARY QUALITY CRITERIA

Interpretability	Completeness
Reliability	Timeliness
Reputation	Ease to use
Credibility	Accessibility
Usefulness	Cost
Added Value	

B. Types of data quality assessment

Objective assessment may use metrics with no consideration of the context application, or may use task dependent metrics, which include the organization's business rules, regulations, and constraints provided by the database administrator, to be applied to any data set [12].

Cleansing techniques: In order to correct, standardize and consequently, to improve data quality, data cleansing has emerged to define and determine error types, search and identify error instances, and correct the errors. "Data cleansing is applied especially when several databases are merged. Records referring to the same entity are represented in different formats in different data sets or are represented erroneously. Thus, duplicated records will appear in the merged database. This problem is known as

merge/purge problem.” [21].

According to [20] the most common methods utilized for error detection are:

- a) Statistical methods through standard deviation, quartile ranges, regression analysis, etc. [18], [19].
- b) Clustering that is a data mining method to classify data in groups to identify discrepancies [25].
- c) Pattern recognition based methods to identify records that do not fit into a certain specific pattern [25].
- d) Association rules to find dependencies between values in a record [21].

Performing Data cleansing in offline time is unacceptable for operational systems. Therefore, cleansing is often regarded as a pre-processing step for Knowledge Discovery in Databases and Data Mining systems during the Extraction Transformation and Load (ETL) process. However, it is still a very time consuming task, “The process of data cleansing is computationally expensive on very large data sets and thus it was almost impossible to do with old technology” [21].

The Parsing technique: By considering the actual data or a metadata, it is possible to determine if a given string (in this case an entire tuple or an attribute) is an element of the language defined by the grammar. Accuracy is commonly assessed by this method.

The assessment of value consistency is calculated objectively by parsing or cleansing techniques.

Sampling: Samples of data are considered appropriate for finding the score of the entire data source. This method is often used for completeness, and accuracy criteria.

Continuous assessment: In case of dynamic criteria, quality assessment is executed at regular intervals. Continuous assessment is required for timeliness, response time, and availability criteria.

Subjective assessment depends upon the user experience, the task at hand, and the use of questionnaires [7], [19].

User experience: Data quality is assessed depending on previous user experience and knowledge of the specific domain and data sources. For instance, reputation and believability are criteria suitable to be judged by user experience assessment.

User sampling: A user will assess data by analyzing several sample results. The user should be skilled enough to find appropriate and representative samples. In the case of interpretability of data, users find which attributes are more suitable for sampling than others are.

Continuous user assessment: In the case where finding representative samples of data is not possible, the user needs to analyze every data, not only samples. That is the case of relevancy or amount of data.

Contract: The assessment is performed depending on the terms of the contract of agreement between the provider and the data consumer, which is the case of price or cost of data [23].

One example of subjective assessment is the use of

control matrices proposed by E. Pierce in [Pierce04], to audit the information products. The evaluation is in terms of how well they meet the consumer’s needs, how well they produce information products, and how well they manage the life cycle of the data after it is produced. The information product manager shall perform the evaluation.

The columns of the control matrix utilized by E. Pierce are the list of data quality problems and the rows correspond to the quality checks or corrective process exercised during the information manufacturing process to prevent them. Each cell shall contain a rating that can have three different forms:

- a) The values Yes or No, whether the quality check exists or not.
- b) The category of effectiveness at error prevention, detection, or correction ranked as “low”, “moderate”, or “high”.
- c) A number to describe the overall level of assessment of the quality check its effectiveness.

From the objective assessment perspective, the original DQM prototype had implemented the parsing technique, sampling and continuous assessment. In the case of subjective assessment this proposal is aimed for use in questionnaires.

III. ASSESSMENT CONSIDERING DATA QUALITY INTERDEPENDENCIES

The present Section is aimed to describe the data quality interdependencies and how they are utilized as part of a new subjective assessment as part of a novelty of the proposal.

A. DQ interdependencies

The assessment of secondary quality criteria relies on data quality interdependencies by the scores aggregation of its components, which in turn might be a primary and/or a secondary quality criterion.

From the Data Quality Reference Model presented in [2], [5], we have identified a number of criteria whose measurement does not depend on other quality criteria as Primary Quality Criteria. Here we present very briefly the following data quality property definitions.

Accuracy: “A measure of the degree of agreement between a data value or collection of data values and a source that has been agreed as being correct.” [9].

Timeliness Is the extent to which the age of data is appropriate for the task at hand [6], and is computed in terms of currency and volatility.

Currency Time interval between the latest update of a data value and the time it is used [14].

The measurement of a quality criterion might be part of the measurement of an aggregate one. The quality dimensions, whose measurements derive from primary criteria, are identified as secondary quality properties.

Completeness is the extent to which data is not missing [12], [23], it is divided by two quality dimensions coverage, and density in [11].

The *interpretability* dimension is the extent to which data are in appropriate language and units, and the data definitions are clear [15]. Thus, it depends on several factors: If there is any change on user needs, its representation should not be affected, this can be possible with a flexible format; The data value shall be presented consistently through the application and that the format is sufficient to represent what is needed and in the proper manner.

Reputation is the extent to which data are trusted or highly regarded in terms of their source or content [12]. Three factors shall be considered at measuring time: reputation of data should be determined by its overall quality. If authors of data provide inaccurate data then they are unreliable and their reputation shall therefore be decreased. Commonly reputation might be increased if data producers have enough experience gained across the time. For instance, when data owners produce accurate data consistently, modify data as soon as possible when mistakes are found, and they in turn recommend data producers of excellent quality data.

Accessibility is the extent to which data is accessible in terms of security [14], availability and cost.

Data might be available but inaccessible for security purposes, or data might be available but expensive.

Data is *credible* as true [12] if it is correct, complete, and consistent.

Usability is the extent to which data are used for the task at a hand with acceptable effort. In other words, users prefer data that are useful and ease to use.

Usefulness is the degree where using data provides benefit on the job performance. In other words, the extent to which users believe data are correct, relevant, complete, timely, and provide added value.

Easy to use is the degree of effort users need to apply to use data [13]. This effort is in terms of understand ability and interpretability as the resources needed to achieve the expected goals. However, it is common that users use determined data sources, due to the reputation of data producers. The measurement of usability allows users to decide on the acceptance of data, and select a specific datum, data or data source among other alternatives.

Data is *reliable* if it is considered as unbiased, good reputation [15] and credible [7].

The *value-added* is stated in terms of how easy it is to get the task completed, also named as effectiveness; how long could the task take known as efficiency; and the personal satisfaction obtained from using data [23].

B. Assessment & Measurement Model

There are some interdependencies very straight forward to compute. For instance, we have already mentioned that

timeliness is the extent to which the age of data is appropriate for the task at hand [6]. Therefore it shall be computed in terms of currency and volatility and fused with an aggregation function as presented in Table 3.

Table 3 MEASUREMENT OF TIMELINESS

Currency	Volatility	Timeliness
$Cu(t)=\text{Time Request} - \text{last update time}$	$Vo(t)= \text{Update frequency}$	$T(t)= \max(0, 1- Cu(t)/Vo(t))$

Furthermore, interpretability is assessed in terms of representation consistency, data appropriateness, data precision, and format flexibility. However, we have not established or tested any kind of correlation among them. Consequently, the questionnaire will consider these 4 criteria, it will ask user to choose from 5 possible answers to identify how consistent, appropriate, etc., is the information he or she utilizes.

The formula to assess interpretability considering the subjective criteria already mentioned is shown in Table 4.

Table 4 MEASUREMENT OF INTERPRETABILITY

Possible answer per each criterion a) Representation Consistency, b) Appropriateness, c) Precision, d) Format Flexibility	Formula to compute Interpretability based on 4 answers a, b, c and d.
Possible answer Very Rarely 0 Rarely 25 Occasionally 50 Frequently 75 Very Frequently 100	Interpretability= $(a+b+c+d)(0.25)\%$
Answers: a, b, c, d	

In the case of reliability, this quality criterion depends on credibility, reputation and unbiased data. However, credibility is measured in terms of objective criteria such as accuracy, completeness and consistency. These last two criteria can be measured directly from data by sql queries or by asking user, the decision is up to the user. Once computed credibility in terms of accuracy, completeness and consistency (computed value a), the second step is to compute reputation which in turn is also in terms of further quality criteria (computed value b). The third step is to compute unbiased data (value c). For instance, the question to obtain “unbiased data” would be: “Is the information unbiased enough to believe the decisions made from it would be reliable?” which possible answer would be yes or no. Yes is interpreted as 0 points and Yes as 100 points.

We are considering a conservative estimation of the secondary quality criteria, the final measure will be the average of its components. Please refer to Table 5 for the corresponding formula to assess reliability.

Table 5 MEASUREMENT OF RELIABILITY

Credibility	Reputation	Unbiased	Reliability
(%TotalAccuracy+ %TotalCompleteness+ %TotalConsistency+) *0.33 OR %TotalCredibility	%TotalReputation	Question/ Answer Yes 100 No 0	(a+b+c)(0.33)
Answer: a	Answer: b	Answer: c	Total: %

Fig. 1 presents the data quality interdependencies. These dependencies can help the measurement of such quality properties.

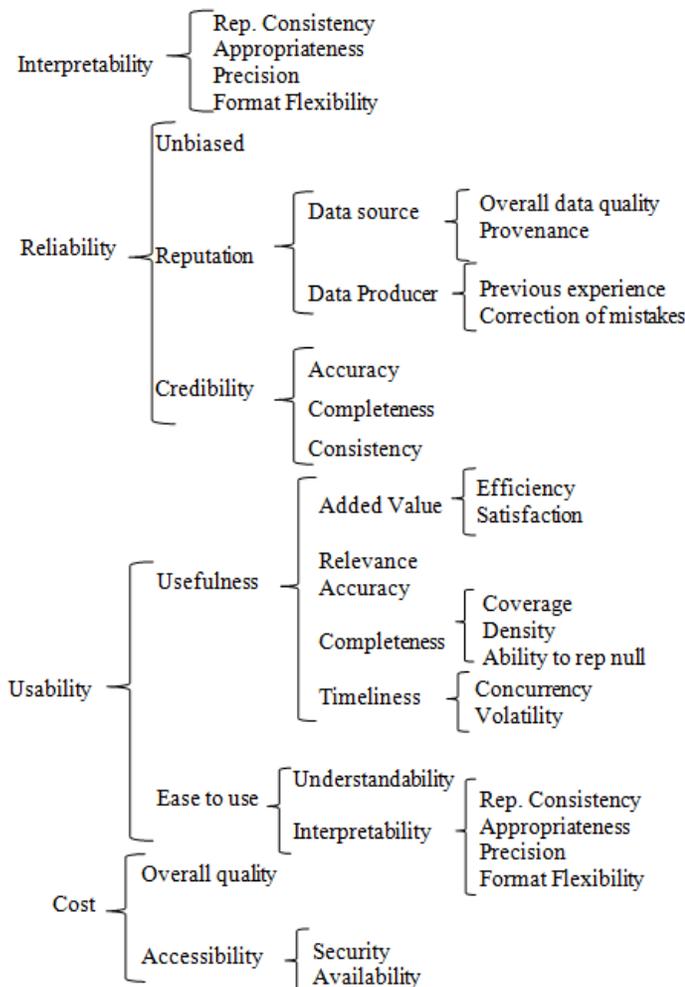


Figure 1. Data Quality Interdependencies

In order to improve the Data Quality Assessment Model, there are four scenarios:

- a) *The objective assessment of primary quality criteria* had been considered within the implementation of the original Data Quality Manager for accuracy,

response time, currency, uniqueness and volatility.

- b) *The objective assessment of secondary quality criteria* had been implemented within the original DQM in the case of completeness, and timeliness.
- c) *The subjective assessment of primary quality criteria* will be considered within the enhanced DQM in the case of the 15 properties.
- d) *The subjective assessment of secondary quality criteria* will be considered in the case of the 11 properties and their interdependencies.

The last two scenarios have been implemented and will be presented in Section IV.

IV. QUESTIONNAIRE FOR THE SUBJECTIVE ASSESSMENT OF DATA QUALITY

This Section presents a questionnaire for the subjective assessment of some quality properties considering their interdependencies.

In the case of subjective quality criteria such as interpretability, credibility, reputation, representation consistency, reliability, added value, usability, usefulness, ease to use and understandability there is not a practical possibility to assess them directly through SQL-queries but by asking expert users only.

We have designed an on-line questionnaire that can adapt questions according to the most relevant quality criteria and their corresponding interdependencies.

This Section presents the questionnaires developed according to the quality interdependencies for those subjective quality criteria. In the case of expert users, the questionnaire requires what quality criteria user wants to assess according to his or her experience, otherwise the user stereotypes are presented. If the criterion depends on other criteria, then a number of questions are presented to the user in order to measure them and to assess the desired quality criteria. Fig. 2 shows the case of interpretability.

*** Required**

WOULD YOU LIKE TO ASSESS DATA INTERPRETABILITY? *

YES

NO

Figure 2. Asking to assess Interpretability

As we have indicated in the previous Section, interpretability relies on representation consistency, appropriateness of data, precision of data and format flexibility. Therefore, five questions are presented to user in order to measure all of them. The possible answers are: very rarely, rarely, occasionally, frequently, very frequently, whereas each of these answers are mapped to a numeric value. Fig. 3 shows the questionnaire for interpretability.

Is information represented consistently throughout the whole application?

Very Rarely
 Rarely
 Occasionally
 Frequently
 Very Frequently

Is the information appropriate for the task at a hand?

Very Rarely
 Rarely
 Occasionally
 Frequently
 Very Frequently

Is the level of data precision enough for a good interpretation of information?

Very Rarely
 Rarely
 Occasionally
 Frequently
 Very Frequently

Is the data format flexible enough to adjust application changes?

Very Rarely
 Rarely
 Occasionally
 Frequently
 Very Frequently

Figure 3. Interpretability Questionnaire

After the questionnaire is completed, the data quality criteria measures are computed and considered within de DQM for the assessment of data quality, or showed as bar graphs at user request. Fig. 4 shows the corresponding bar graph.

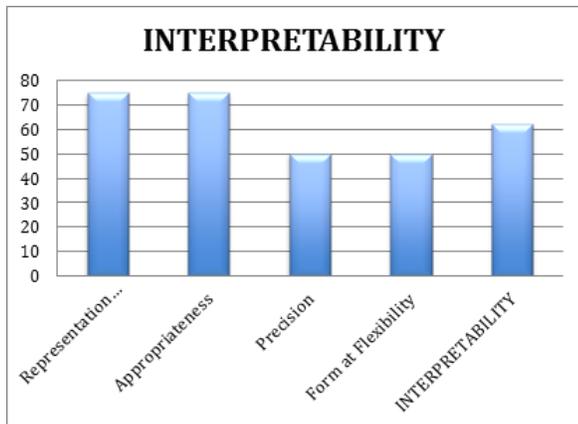


Figure 4. Interpretability assessment as percentage

The assessment of reliability is computed based on unbiased data, credibility and reputation. However credibility and reputation are secondary criteria, credibility measurement depends on completeness, accuracy, and consistency.

Reputation depends on what user trust the most: data source, data provider or both. Therefore, the questionnaire asks the user the corresponding quality properties. Fig. 5 shows the reliability questionnaire.

RELIABILITY

Is the information unbiased enough to believe the decisions made from it would be reliable? *

Reliability - Unbiased

Yes
 No

How often is data accurate? *

Credibility - Accuracy

Most of the time
 Some of the time
 Hardly ever
 Very seldom

Is the information consistent in terms of business rules and data values. *

Credibility - Consistency

Most of the time
 Some of the time
 Hardly ever
 Very seldom

Is the information complete enough in terms of the records you need for the task at a hand? *

Completeness - Density

Most of the time
 Some of the time
 Hardly ever

Is the information complete enough in terms of the attributes you need for the task at a hand? *

Completeness - Coverage

Most of the time
 Some of the time
 Hardly ever
 Very seldom

Figure 5. Reliability Questionnaire

The reliability questionnaire is composed of five questions: the first one is relative to unbiased data, the following two questions are regarding credibility of data and the remaining questions are focused on completeness. See Fig. 6 for the corresponding assessment.



Figure 6. Reliability assessment as percentage

According to his or her answers, data consumer can observe from Fig. 6 an 83% of reliability derived from completely reliable data, 80% of credible data because sometimes data is not complete or accurate and the correction of mistakes are not always on time. Due to space restrictions, the present paper is not showing completeness, accuracy and consistency measures for credibility neither measures for reputation.

V. USER PROTOTYPES

The identification and ranking of relevance for data quality properties according to the type of users and Information Systems is not straightforward. For instance, if we consider volatility as the update frequency the relevance of such quality property varies very remarkable according to the application domain, volatility is essential within operational systems, but not quite important within DSS where historical information is materialized.

An Executive Support System (ESS) is designed to help a senior management tackle and address issues and long-term trends to make strategic decisions for the business. It gathers analyses and summarizes aggregate, internal and external data to generate projections and responses to queries. Therefore, the main data quality problem on ESS relies on external data, so decisions depend on accuracy, timeliness, completeness and currency of the external data collected. Furthermore, users are interested in those quality properties that are very much related to their work role.

According to Lee and Strong [10], the responses from data collector, data custodian, and data consumer within the data production process determine data quality because of their knowledge. Data consumers require friendly and usable tools in order to deal with making decisions only rather than the IS per se. Possible inconsistencies might be derived from different data sources so making decisions regarding which external data source to trust is an issue. Response time however, is not of great relevance when the analysis is on long-term trends.

A. Data Collector in DSS

Within a Decision Support System, there are people, groups or even systems that generate, gather or save data to the information systems. Consequently, data collectors impact on accuracy, completeness, currency and timeliness of data. The quality properties identified as the most relevant within Decision Support Systems for data collectors are presented in Fig. 7.

Accuracy, completeness and timeliness shall be presented to the collector user in order to help during the assessment of data quality. Furthermore, completeness is estimated by an aggregation function of coverage, density and ability to represent nulls. Same applies for the rest of the user stereotypes.

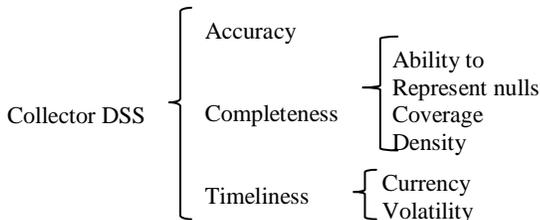


Figure 7. Quality properties for collectors within DSS

B. Data Custodian in DSS

Data custodians are people who manage computing resources for storing and processing data.

In the case of DSS, the process of extraction, transformation and load (ETL) of data within a data warehouse is mainly related to data custodians.

The ETL process is a key data quality factor; it may degrade or increase the level of quality. Therefore, custodians determine the representation of data, value consistency, format precision, appropriateness of data for the task at a hand, the efficient use of storage media.

In other words, appropriateness, concise representation, efficient use of storage media, format precision, representation consistency and value consistency shall be evaluated and presented to them in order to help them decide which data source should be utilized.

Fig. 8 is presented for the relevant quality properties among data custodians within Decision Support Systems.

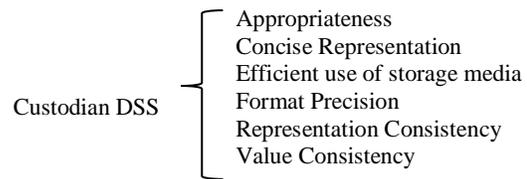


Figure 8. Quality properties for custodians within DSS

C. Data Consumer in DSS

Data consumers are involved in retrieval of data, additional data aggregation and integration. Therefore, they have an impact on accuracy, amount of data relevant for the task at a hand, usability, accessibility, reliability and cost of information in order to make decisions. An analysis on data quality properties in Data warehouses is presented in [8]; such quality properties are included in this work. Accuracy, amount of data, usability, accessibility, reliability and cost shall be considered during data quality assessment. Such quality properties are shown in Fig. 9.

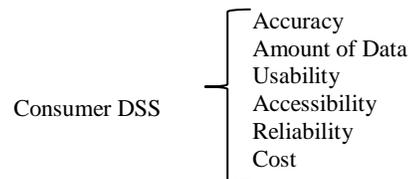
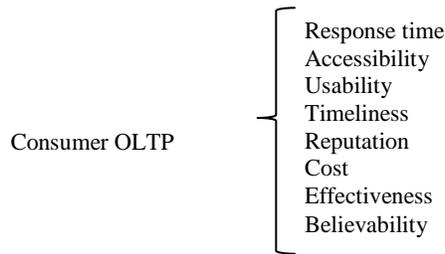


Figure 9. Quality Properties for data consumer within DSS

D. Data Consumer OLTP

As data consumers are involved in retrieval of data the quality properties usability, accessibility, believability, reputation of data sources are key factors for their job. Response time and timeliness [14] are essential within

OLTP systems. From the data consumer perspective accessibility [15] and cost are also very important. The



corresponding quality properties relevant to this role are shown in Fig. 10.

Figure 10. Quality properties for consumers within OLTP systems

E. Data Custodian in OLTP

In transactional systems, data custodians are much related to accuracy, consistency at data value level, completeness [9], timeliness [13], and uniqueness. Therefore the set of quality properties they are interested on for analysis of data quality from their perspective are shown in Fig. 11.

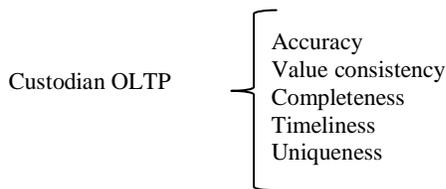


Figure 11. Quality properties for custodian within OLTP systems

F. Data Collector in OLTP

As data collectors within OLTP systems are people who generate information, this role impacts on accuracy, completeness, currency, uniqueness, value consistency and volatility of data. Fig. 12 presents such relevant quality properties for collectors within OLTP systems.

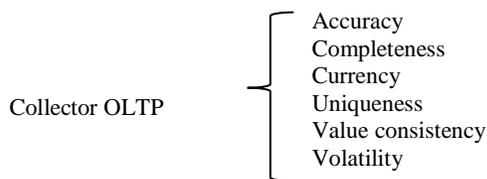


Figure 12. Quality properties for collector within OLTP systems

A number of quality properties have been identified according to the type of users and shown in the past 6 figures. However, there are no priorities assigned to such quality properties in order to assign a weight for assessment purposes.

The following Section shows an example of the online survey developed to provide such ranking.

VI. A SURVEY FOR RANKING DATA QUALITY PROPERTIES WITHIN THE USER PROTOTYPES

This Section presents a survey that was conducted to provide a ranking of quality properties according to the type of Information System from experienced users.

A. Design of Questionnaire

As user experience is substantial within the data quality assessment, a survey was applied to OLTP and OLAP specialists on the web. Therefore, we have conducted an on-line survey requesting an order of importance among the quality properties according to their corresponding experience within a specific Information System. The questionnaire requires the type of information system and what role do users play. According to these two characteristics, the questionnaire presents a set of quality properties and a percentage of relevance these properties should be assigned during quality assessment.

In order to obtain unbiased results, we have invited a number of specialists in operational and DSS information systems around the world. The following groups were invited to participate within the survey:

- a) *The University Network of Contribution in Software Engineering and Databases:* To take into account a specialized academic perspective.
- b) *The Professionals of Business Intelligence Group:* To retrieve all the experience from a very pragmatic perspective involved in Business Intelligence.
- c) *The Very Large Database Group:* In order to obtain the perspective and experience from the databases group.
- d) *The Data Quality Pro Group:* The retrieval of the relevance of data quality properties from data quality experts is part of our proposal.
- e) *The Information Technology and Communications Group:* For obtaining a broad overview within the Information Technology perspective of the relevant of quality priorities from this group.

Experienced people from these groups have answered our survey, and have established which quality properties are more relevant and under which hierarchy, considering their role and the type of information system in which they are involved.

The questionnaire was designed to be briefly answered, and it was available for six months in order to allow these experts the specification of those quality priorities within the analysis of data quality. For instance, we present in Fig. 13 an example of the one developed to find out the relevance of quality properties for custodian users within OLTP systems.

B. Results of Experiments

Quality criteria identification

Select the type of information System you are involved in:

On-line Transaction Processing (OLTP)

What type of IT user are you?

Collector Custodian Consumer

According to your role, choose from the following criteria those more relevant for assessing data quality.

Quality Properties	Usage rate (%)
Accuracy	60
Value Consistency	40
Completeness	60
Timeliness	60
Uniqueness	40

Send

Figure 13. On-line survey for custodian users within OLTP systems

Quality is a very subjective concept; it depends on user experience, information system, business sector, among other factors. As a consequence, we have decided to collect expert opinions in order to identify the most common ranking of such quality properties they utilized during data quality assessment. The results of the on-line survey were analyzed and are presented in this Section.

There were 136 responses collected. Regarding Decision Support Systems 82 DSS specialists participated and expressed their opinion. 22 of them were user collectors, 33 data custodians, and 27 DSS consumers.

According to data collectors, accuracy, completeness, currency and timeliness are the most relevant quality properties to take into account during quality assessment on Decision Support Systems. Accuracy and completeness are equally important with 30 percent each followed by currency and timeliness with 20 percent.

Data custodians considered as first option currency and volatility of data with 30 percent, followed by accuracy, completeness, uniqueness and value consistency with 10 percent each.

Data consumers on the other hand, rely a total of 60 percent on accuracy, amount of data and usability to make their decisions regarding data quality, followed by reliability, easy to use, interpretability and relevance. Refer to Fig. 14 for the data quality prioritization within DSS.

In the case of Online Transaction Processing Systems 54 specialists have participated and answered our online survey, 19 of them were user collectors, 13 data custodians, and 22 data consumers.

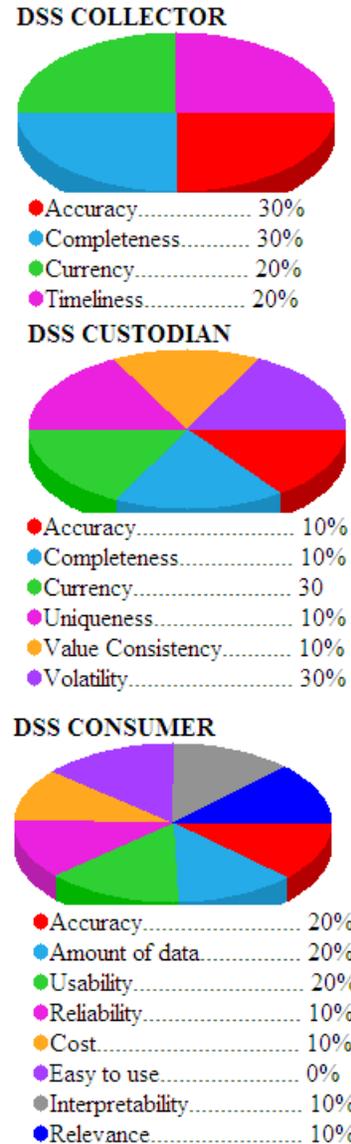


Figure 14. Data Quality prioritization within DSS

On the one hand, data collectors prefer complete data with 30 percent, accurate, and non-duplicated data with 20 percent each, followed by current and consistent information with 10 percent each. Data custodians also trust accurate, unique data with 30 percent each followed by complete data rather than timely data.

On the other hand, data consumers require fast response time, accessible, timely and usable data.

Refer to Fig. 15 for the corresponding data quality prioritization.

The final percentages are obtained through the ranking of the quality properties according to the responses collected.



Figure15. Data Quality prioritization within OLTP systems

The present research is looking forward to having more responses in the future by incorporating more specialist groups that allow being more precise with the outcomes and also to test the effectiveness of the stereotypes presented.

In the case of data quality interdependencies, we have taken a conservative approach by computing the average of the components in order to obtain the overall quality measure of an specific secondary data quality property, we have no identified any correlation among them, this correlation is part of our future work.

The following Section presents the Data Quality Manager we have improved by taking into consideration the subjective assessment from experts by the specification

of such quality priorities within a weighted matrix during the assessment process carried out by the execution of ranking and scaling methods.

VII. DATA QUALITY MANAGER IMPROVEMENT

We have developed a Data Quality Manager as a prototype for the assessment of data quality within heterogeneous databases in [1], [2], [3], [4], in the case of quantitative or primary data quality criteria, the assessment is performed by SQL queries or by the validation of implementation of integrity constraints.

An improvement of such prototype consisted in the implementation of the data quality stereotypes to be suggested to inexperienced users to assist them with the analysis of a number of data sources to identify and query the best ranked data sources and make informed decisions.

The stereotypes implemented are the result of the experiments conducted through the analysis of the results obtained from the online survey and briefly explained in the previous Section.

A. Suggestion of priorities for quality priorities according to the information systems

This Section presents very briefly the improvement of the DQM prototype for the assessment of data quality by suggesting a set of quality properties and their priorities to naive users. In the case of experienced users they still allowed to indicate explicitly their preferences.

For instance, Fig. 16 shows the DQM main menu and the selection of data quality assessment within Online Transaction Processing System conducted by non-expert custodian user.



Figure 16. DQM main menu for selecting IS and type of user

Fig. 17 shows how the DQM prototype suggest a non-expert custodian user the most relevant quality properties within an OLTP system, such as accuracy, value consistency, completeness, timeliness, uniqueness and their corresponding priorities according to our expert users. For instance, accuracy and uniqueness are the most relevant quality properties with 30%, then completeness with 20%, finally, timeliness, and value consistency with 10%.

The following Section is aimed to briefly summarize the assessment of data quality. For further information, please refer to [2], [3].

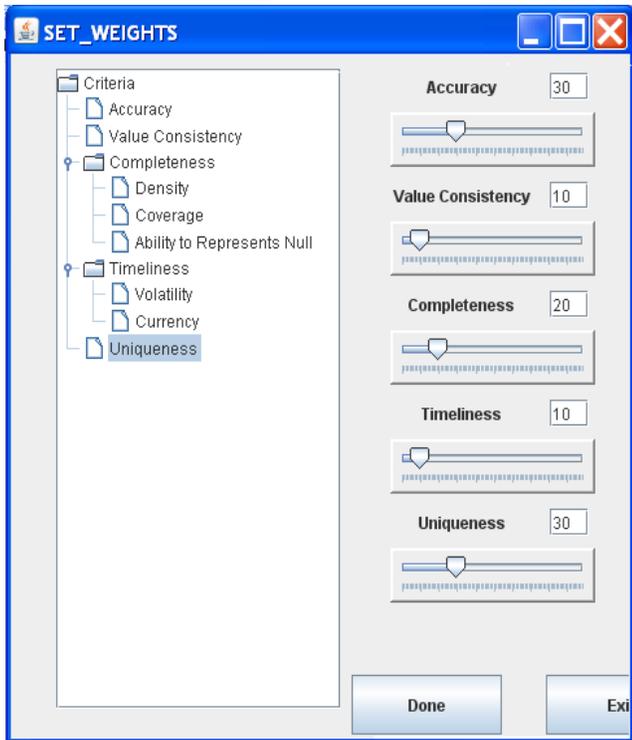


Figure 17. DSS most relevant quality properties

B. Assessment of Data Quality

Having all the quality properties prioritized, the weights are normalized. The next step within The DQM prototype is the selection of the data sources, scaling, and ranking methods.

In order to select data sources from a scroll pane, the prototype retrieves from the metadata all the data sources involved in the federation of interest.

The scaling method is selected by pressing its corresponding radio button and the ranking of data sources is executed by pressing the buttons TOPSIS or SAW.

Fig. 18 shows the assessment of data quality properties with Norma and SAW methods respectively.

Please refer to [2], [3] for further information. The overall quality is presented in descendent order in a Text area.

There are three ranked data sources shown in Fig. 18, which were obtained from the TPCC benchmark [24] named TPCCA, TPCCB and TPCCD, where TPCCD contains the best overall data quality.

We have validated the DQM prototype against the specification of the model, and we have verified that the Data Quality Manager (DQM) can provide appropriate information about the qualitative nature of the data been returned from the data sources.

Section VIII presents conclusion and future work.

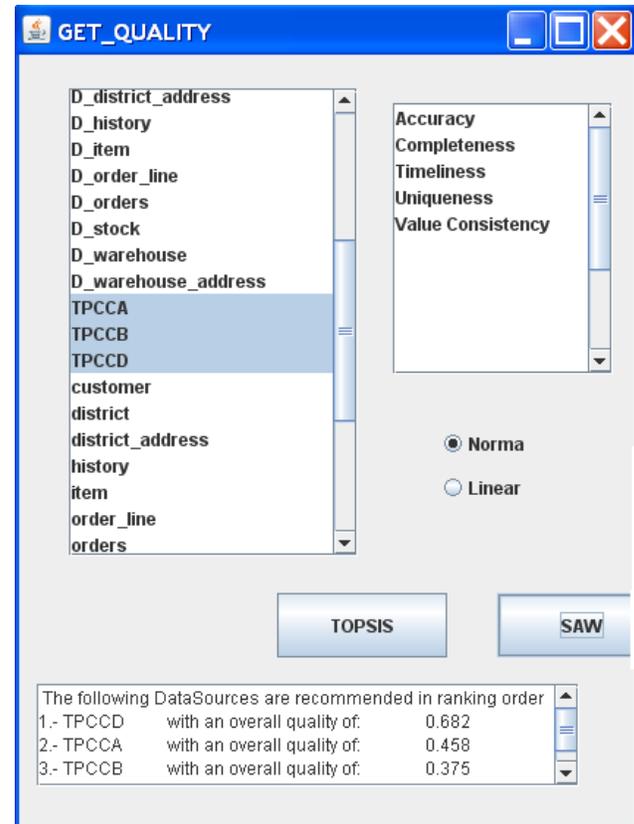


Figure 18. Ranking of data sources according to their quality

VIII. CONCLUSION AND FUTURE WORK

Nowadays data quality has been considered one of the most important factors for making business, especially in terms of profitability and competitive advantage. There are several factors that can affect data quality. Previous research has been conducted in order to propose a data quality assessment framework on the bases of a Reference Model, Measurement Model, and an Assessment Model. These models have been implemented within a Data Quality Manager Prototype, such original implementation was focused only on two out of four scenarios, considering objective assessment of primary criteria, very few secondary criteria where addressed, mainly focused on expert users. The purpose of the present research has been to establish an extended assessment framework and a data quality assessment tool that can help non-expert users according to the type information system and to implement the assessment of subjective quality properties to provide more information to expert users.

The Reference Model has been extended by identifying a set of relevant quality properties according to the type of Information Systems (IS) and the role of user, named as user stereotypes [6].

The Measurement Model has been extended in order to provide specific metrics for subjective quality criteria

considering the interdependencies among them.

The Assessment Model has been extended by identifying a set of data quality priorities from user prototypes, to establish their corresponding weights during the assessment process [1].

The Assessment Model has been enhanced by identifying a subjective assessment method to incorporate secondary quality properties.

The new Reference, Measurement and Assessment Models have been implemented within the original Data Quality Manager Prototype. Therefore, the DQM is now able to suggest a set of ranked data quality properties to inexperienced users to assist them with the analysis of a number of data sources to query the best ad-hoc ranked data sources to support informed decision making.

The Data Quality Manager is able now to assess automatically, semi automatically or manually. However, more information from specialists is required in order to corroborate the prioritization and testing of the effectiveness of the stereotypes identified. This further feedback from the specialists is part of future work.

There are some quality properties whose measurement and assessment methods are suitable to be enhanced as is the case of accuracy and uniqueness. The incorporation of data mining techniques is also part of future work.

In the case of data quality interdependencies, we have taken a conservative approach by computing the average of the components in order to obtain the overall quality measure of an specific secondary data quality property, we have no identified any correlation among them, this correlation is part of our future work.

The present research is part of an effort to improve data quality in order to help users to make business by taking informed decisions. Consequently, after performing subjective and objective data quality assessments, comparing the results of the assessments, is important to identify discrepancies, and to determine root causes of errors for determining and taking necessary actions for improvement.

REFERENCES

- [1] P. Angeles and F. Garcia-Ugalde, "Relevance of quality criteria according to the type of information systems", Proc. International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 12), Mar. 2012, pp. 175-181, ISBN: 978-1-61208-185-4.
- [2] P. Angeles and L.M. MacKinnon, "Managing Data Quality of integrated data with Known Provenance", International Journal of Information Quality, ISSN (Online): 1751-0465, ISSN (Print): 1751-0457, Vol.2 No. 3, 2011.
- [3] P. Angeles and F. Garcia-Ugalde, "Assessing data quality of integrated data by quality aggregation of its ancestors", *Computación y Sistemas*, Vol. 13 No. 4, ISSN 1405-5546.
- [4] P. Angeles and F. Garcia-Ugalde, "Assessing quality of derived non atomic data by considering conflict resolution function", Proc. International Conference on Advances in Databases (DBKDA 09), Mar. 2009, pp. 81-86, IEEE Computer Society, ISBN: 978-0-7695-3550-0 doi>10.1109/DBKDA.2009
- [5] P. Angeles and F. Garcia-Ugalde, "A data quality practical approach, International Journal On Advances in Software, Vol. 2, No. 3, 2009, pp. 259-274, ISSN 1942-2628.
- [6] P. Angeles and F. Garcia-Ugalde, "User stereotypes for the analysis of data quality according to the type of information systems", Proc. International Association for Scientific Knowledge (IASK 08), E-Activity and Leading Technologies, pp. 207-212, Dec. 2008.
- [7] Ballou, D.P., Wang, R.Y., Pazer, H., and Tayi, G.K., "Modeling information manufacturing systems to determine information product quality", *Management Science*, Apr. 1998, pp. 462-484.
- [8] C.Cappiello, C.Francalanci, B.Pernici, P.Plebani, and M.Scannapieco, "Data quality assurance in cooperative information systems: a multi-dimension quality certificate", Proc. International Workshop "Data Quality in Cooperative Information Systems" (DQCIS 2003), pp. 64 -70 Siena, Italy, 2003.
- [9] M. Jarke, M.A. Jeusfeld, C. Quix, and P.Vassiliadis, "Architecture and quality in data warehouses an extended repository approach", *Journal on Information Systems*, Vol. 24", no.3, pp. 229-253, 1999.
- [10] Lee Y. and Strong D. "Knowing-Why about data processes and data quality", *Journal of Management Information Systems*, Vol. 20, No. 3, pp. 13 - 39. 2004.
- [11] F. Naumann, J. Freytag, and U. Lesser, "Completeness of information sources", Workshop on Data Quality in Cooperative Information Systems (DQCIS2003), pp. 583-615, Cambridge, Mass., 2003.
- [12] L. Pipino, W.L. Yang, and R. Wang, "Data quality assessment", *Communications of the ACM*, Vol. 44 no. 4e, pp.211-218, 2002.
- [13] Redman "Data Quality for the Information Age", Boston MA., London: Barteck House, 1996.
- [14] D.M. Strong, W.L. Yang, and R.Y. Wang, "Data quality in context", *Communications of the ACM*, vol. 40, no. 5, pp. 103-110, 1997.
- [15] R.Y. Wang, M.P. Reedy, and A. Gupta, "An object-oriented implementation of quality data products". Workshop on Information Technology Systems, 1993.
- [16] Wang R. Y., and Strong D.M. "Beyond accuracy: What data quality means to data consumers", *Journal of Management of Information Systems*, vol. 12, no 4 1996, pp. 5 -33.
- [17] R. Wang, "A product perspective on total data quality management", *Communications of the ACM*, vol. 41, no. 2, pp.58-65, 1998.
- [18] Barnett, V., and Lewis, T.: 1984, *Outliers in statistical data*, John Wiley & Sons, New York.
- [19] R.K. Bock, and W. Krischer, "The data Analysis brief book" Springer 1998.
- [20] Buchheit R. B., "Vacuum: automated procedures for assessing and cleansing civil infrastructure data", PhD Thesis, May 2002
- [21] Maletic, J. I., and Marcus, "Data cleansing: Beyond integrity checking". Proc. Conference on Information Quality (IQ2000) (Massachusetts Institute of Technology, October 2000), pp. 200-209.
- [22] F. Naumann, and C. Roker C., "Assessment methods for information quality criteria", Proc. International Conference on Information Quality (IQ2000), Cambridge, Mass., 2000.
- [23] F. Naumann, "Quality-Driven query answering for integrated information systems", Lecture Notes in Computer Sciences LNCS 2261, Springer Verlag, Heidelberg, 2002.
- [24] TPCH, TPC Benchmark™ H, Standard Specification Revision 2.3.0 Transaction Processing Performance Council www.tpc.org/info (retrieved: December 2012).
- [25] J. Han and M. Kamber, "Data Mining: Concepts and Techniques", 2nd ed. The Morgan Kaufmann Series in Data Management Systems, Jim Gray, Series Editor Morgan Kaufmann Publishers, 2006. ISBN 1-55860-901-6.

Analyzing 3D Complex Urban Environments Using a Unified Visibility Algorithm

Oren Gal

Mapping and Geo-information Engineering
Technion - Israel Institute of Technology
Haifa, Israel
e-mail: orengal@technion.ac.il

Yerach Doytsher

Mapping and Geo-information Engineering
Technion - Israel Institute of Technology
Haifa, Israel
e-mail: doytsher@technion.ac.il

Abstract - This paper presents a unique solution for the visibility problem in 3D urban environments. We shall introduce a visibility algorithm for a 3D urban environment, based on an analytic solution for basic building structures. A building structure is presented as a continuous parameterization approximating of the building's corners. The algorithm quickly generates the visible surfaces' boundary of a single building. Using simple geometric operations of projections and intersections between visible pyramid volumes, hidden surfaces between buildings are rapidly computed. Furthermore, extended visibility analysis for complex urban environments, consisting of mass modeling shapes, is presented. Mass modeling consists of basic shape vocabulary with a box as the basic structure. Using boxes as simple mass model shapes, one can generate complex urban building blocks such as *L*, *H*, *U*, and *T* shapes. The visibility analysis is based on concatenating the analytic solution for the basic single box building structure. The algorithm, demonstrated with a schematic structure of an urban environment and compared to the Line of Sight (LOS) method, demonstrates the computation time efficiency. Real urban environment approximated to the 3D basic shape vocabulary model demonstrates our approach.

Keywords - *Visibility; 3D; Urban environment; Spatial analysis; Mass modeling*

I. INTRODUCTION

In the past few years, the 3D GIS domain has developed rapidly, and has become increasingly accessible to different disciplines. Spatial analysis in a 3D environment appears to be one of the most challenging topics in the communities currently dealing with spatial data. One of the most basic problems in spatial analysis is related to visibility computation in such an environment. Visibility calculation methods aim to identify the parts visible from a single point, or multiple points, of objects in the environment.

The visibility problem has been extensively studied over the past twenty years, due to the importance of visibility in GIS, computer graphics, computer vision and robotics. Most previous works approximate the visible parts to find a fast solution in open terrains, and do not challenge or suggest solutions for a dense urban environment. The exact visibility methods are highly complex, and cannot be used for fast

applications due to the long computation time. Gal and Doytsher [1] recently presented a fast and exact solution for the 3D visibility problem in urban environments based on an analytic solution. Other fast algorithms are based on the conservative Potentially Visible Set (PVS) [2]. These methods are not always completely accurate, as they may include hidden objects' parts as visible due to various simplifications and heuristics.

In this paper, we introduce a new, fast and exact solution for the 3D visibility problem from a viewpoint in an urban environment, which does not suffer from approximations. We consider a 3D urban environment building modeled as a cube (3D box) and present an analytic solution for the visibility problem. The algorithm computes the exact visible and hidden parts from a viewpoint in an urban environment, using an analytic solution, without the expensive computational process of scanning all objects' points. The algorithm is demonstrated by a schematic structure of an urban environment, which can also be modified for other complicated urban environments, with simple topological geometric operators. In such cases, computation time grows almost linearly.

Our method uses simple geometric relations between the objects and the lines connecting the viewpoint, and the objects' boundaries by extending the visibility boundary calculation from 2D to a 3D environment, using approximated singular points [3]. The spatial relationship between the different objects is computed by using fast visible pyramid volumes from the viewpoint, projected to the occluded buildings.

Based on our visibility solution [1], we extend our research and introduce a fast and exact solution to the 3D visibility problem in complex urban environments, generated by mass modeling shapes and a procedural modeling method. Our solution can be carried out in a near Real Time performance. We consider a 3D urban environment, which can be generated by grammar rules. The basic entities are basic vocabulary mass modeling, such as *L*, *H*, *T* profile shapes that can be separated into simple boxes. Based on our visibility method, we analyze the spatial relations for each profile and compute the visible and the hidden parts. Each box is a basic building modeled as 3D cubic parameterization, which enables us to implement an analytic

solution for the visibility problem, without the expensive computational process of scanning all the objects' points.

The algorithm is demonstrated by a collection of basic mass modeling shapes of an urban environment, where each shape can be sub-divided into a number of boxes. Using an extension of our analytic solution for the visibility problem of a single box from a viewpoint, an efficient solution for a complex environment is demonstrated. We also compared computation time between the presented method and the traditional "Line of Sight" (LOS) method.

II. RELATEDWORK

Accurate visibility computation in 3D environments is a very complicated task demanding a high computational effort, which could hardly have been performed in a very short time using traditional well-known visibility methods [4], [5]. Previous research in visibility computation has been devoted to open environments using DEM models, representing raster data in 2.5D (Polyhedral model). Most of these works have focused on approximate visibility computation, enabling fast results using interpolations of visibility values between points, calculating point visibility with the LOS method [6], [7].

A vast number of algorithms have been suggested for speeding up the process and reducing the computation time [8]. Franklin [9] evaluates and approximates visibility for each cell in a DEM model based on greedy algorithms. An application for sitting multiple observers on terrain for optimal visibility cover was introduced in [10]. Wang et al. [11], introduced a Grid-based DEM method using viewshed horizon, saving computation time based on relations between surfaces and Line Of Sight (LOS), using a similar concept of Dead-Zones visibility [12]. Later on, an extended method for viewshed computation was presented, using reference planes rather than sightlines [13].

One of the most efficient methods for DEM visibility computation is based on shadow-casting routine. The routine cast shadowed volumes in the DEM, like a light bubble [14]. Other methods related to urban design environment and open space impact treat abstract visibility analysis in urban environments using DEM, focusing on local areas and approximate openness [15], [16]. Extensive research treated Digital Terrain Models (DTM) in open terrains, mainly Triangulated Irregular Network (TIN) and Regular Square Grid (RSG) structures. Visibility analysis on terrain was classified into point, line and region visibility, and several algorithms were introduced based on horizon computation describing visibility boundary [17], [18].

Only a few works have treated visibility analysis in urban environments. A mathematical model of an urban scene, calculating probabilistic visibility for a given object from a specific viewcell in the scene, has been presented by [19]. This is a very interesting concept, which extends the traditional deterministic visibility concept. Nevertheless, the buildings are modeled as circles, and the main challenges of spatial analysis and building model were not tackled.

Other methods were developed, subject to computer graphics and vision fields, dealing with exact visibility in 3D scenes, without considering environmental constraints.

Plantinga and Dyer [5] used the aspect graph – a graph with all the different views of an object. Shadow boundaries computation is a very popular method, studied by [2],[20],[21]. All of these works are not applicable to a large scene, due to computational complexity.

As mentioned, online visibility analysis is a very complicated task. Recently, off-line visibility analysis, based on preprocessing, was introduced. Cohen-Or et al. [22] used a ray-shooting sample to identify occluded parts. Schaufler et al. [23] use blocker extensions to handle occlusion.

Shape grammars, which are an inherent part of the procedural modeling method, have been used for several applications over the past years. The first and original formulation of shape grammar deals with arrangement and location of points and labeled lines. Therefore, this method was used for architecture applications, for construction and analysis of architectural design [24].

Modeling a 3D urban environment can be done by dividing and simplifying the environment using a set of grammar rules consisting of basic shape vocabulary of mass modeling [25]. By that, one can simply create and analyze 3D complex urban environments using computer implementation.

Automatic generation or modeling of complex 3D environments, such as the urban case, can be a very complicated task dealing with fast computations analysis. In our case, visibility computation in 3D environments is a very complicated task, which can hardly be performed in a very short time using traditional well-known visibility methods, due to the environment's complexity, modeled with or without a procedural modeling method.

III. URBAN ENVIRONMENT MODELING

A. Procedural Modeling

Procedural modeling consists of production rules that iteratively create more and more details. In the context of urban environments, grammar rules first generate crude volumetric models of buildings, named as mass modeling, which will be introduced in the next sub-section. Iterative rules can also be applied to facade windows and doors. Modeling processes of the environment also specify the hierarchical structure.

Shape grammar, which is also called Computer Generated Architecture (CGA) shape, produces buildings' shells in urban environments with high geometric details. A basic set of grammar rules was introduced by Wonka et al. [25].

Procedural modeling enables us to create fast and different three-dimensional urban models using a combination of random numbers and stochastic rule selection with different heights and widths. An example model using these four rules is depicted in Figure 1.

B. Mass Modeling

Modeling urban environments can be a very complicated task. The simplest constructions use boxes as a basic structure. By using boxes as simple mass models, one can

generate basic buildings blocks such as L, H, U, and T shapes, demonstrated in Figure 2.



Figure 1. Generating Urban Environment Using CGA Shape Based on Mass Modeling (source: [26])

An extended mass modeling of roofs and facades for building models was introduced by Muller et al. [26]. In this paper, we introduce visibility analysis of the basic shape vocabulary of mass modeling using a box's basic structure, described as visibility computation of a basic shape vocabulary.

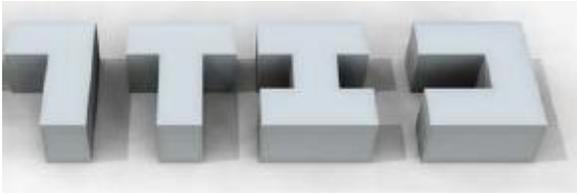


Figure 2. Basic Shape Vocabulary for Mass Modeling (source: [27])

IV. PROBLEM STATEMENT

We consider the basic visibility problem in a 3D urban environment, consisting of 3D buildings modeled as 3D cubic parameterization $\sum_{i=1}^N C_i(x, y, z_{h_{min}}^{h_{max}})$, and viewpoint $V(x_0, y_0, z_0)$.

Given:

- A viewpoint $V(x_0, y_0, z_0)$ in 3D coordinates
- Parameterizations of N objects $\sum_{i=1}^N C_i(x, y, z_{h_{min}}^{h_{max}})$, describing a 3D urban environment model

Computes:

- Set of all visible points in $\sum_{i=1}^N C_i(x, y, z_{h_{min}}^{h_{max}})$ from $V(x_0, y_0, z_0)$.

This problem would appear to be solved by conventional geometric methods, but as mentioned before, this demands a long computation time. We introduce a fast and efficient computation solution for a schematic structure of an urban environment that demonstrates our method.

V. ANALYTIC VISIBILITY COMPUTATION

A. Analytic Solution for a Single Object

In this section, we first introduce the visibility solution from a single point to a single 3D object. This solution is based on an analytic expression, which significantly improves time computation by generating the visibility boundary of the object without the need to scan the entire object's points.

Our analytic solution for a 3D building model is an extension of the visibility chart in 2D introduced by Elber et al. [3] for continuous curves. For such a curve, the silhouette points, i.e. the visibility boundary of the object, can be seen in Figure 3:

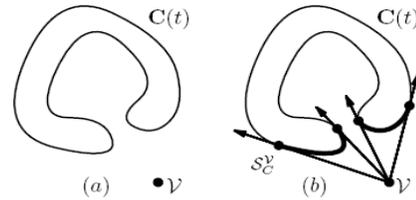


Figure 3. Visible Silhouette Points SCV from viewpoint V to curve $C(t)$ (source: [3])

The visibility chart solution was originally developed for dealing with the Art Gallery Problem for infinite viewpoint; it is limited to 2D continuous curves using multivariate solver [3], and cannot be used for on-line application in a 3D environment.

Based on this concept, we define the visibility problem in a 3D environment for more complex objects as:

$$C'(x, y)_{z_{const}} \times (C(x, y)_{z_{const}} - V(x_0, y_0, z_0)) = 0 \quad (1)$$

Where 3D model parameterization is $C(x, y)_{z_{const}}$, and the viewpoint is given as $V(x_0, y_0, z_0)$. Solutions to equation (1) generate a visibility boundary from the viewpoint to an object, based on basic relations between viewing directions from V to $C(x, y)_{z_{const}}$ using cross-product characters.

A three-dimensional urban environment consists mainly of rectangular buildings, which can hardly be modeled as continuous curves. Moreover, an analytic solution for a single 3D model becomes more complicated due to the higher dimension of the problem, and is not always possible. Object parameterization is therefore a critical issue, allowing us to find an analytic solution and, using that, to generate the visibility boundary very quickly.

1) *3D Building Model*: Most of the common 3D City Models are based on object-oriented topologies, such as 3D Formal Data Structure (3D FDS), Simplified Spatial Model (SSS) and Urban Data Model (UDM) [28]. These models are very efficient for web-oriented applications. However, the fact that a building consists of several different basic features makes it almost impossible to generate analytic representation. A three-dimensional building model should be, on the one hand, simple, enabling analytic solution, and

on the other hand, as accurate as possible. We examined several building object parameterizations, and the preferred candidate was an extended n order sphere coordinates parameterization, even though such a model is a very complex one, and will necessitate a special analytic solution. We introduce a model that can be used for analytic solution of the current problem. The basic building model can be described as:

$$x = t, y = \left(\frac{x^n - 1}{1 - x^n} \right), z = c \quad (2)$$

$$-1 \leq t \leq 1, n = 350, c = c + 1$$

This mathematical model approximates building corners, not as singular points, but as continuous curves. This building model is described by equation (2), with the lower order badly approximating the buildings' corners, as depicted in Figure 4. Corner approximation becomes more accurate using n=350 or higher. This approximation enables us to define an analytic solution to the problem.

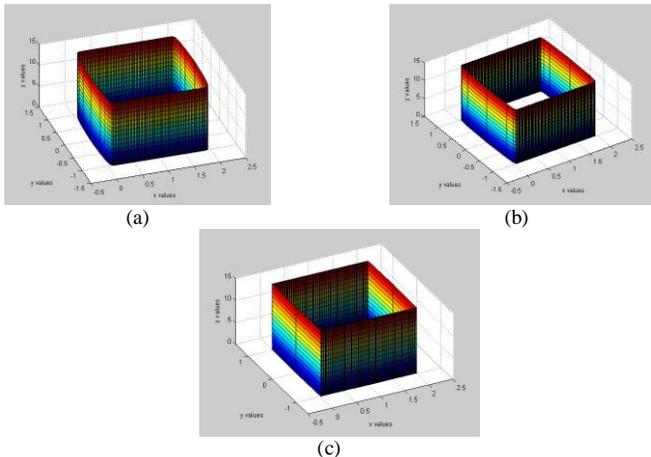


Figure 4. Topside view of the building model using equation (2) - (a) n=50; (b) n=200; (c) n=350

We introduce the basic building structure that can be rotated and extracted using simple matrix operators (Figure 5). Using a rotation matrix does not affect our visibility algorithm, and for a simple demonstration of our method we present samples of parallel buildings.

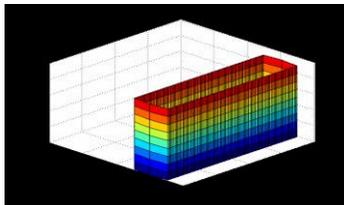


Figure 5. 3D Analytic Building Model with Equation (2), where $Z_{h_{min}}=0$ and $Z_{h_{max}}=9$

2) *Analytic Solution for a Single Building:* In this part we demonstrate the analytic solution for a single 3D building

model. As mentioned above, we should integrate building model parameterization to the visibility statement. After integrating eq. (1) and (2):

$$C'(x, y)_{z_{const}} \times (C(x, y)_{z_{const}} - V(x_0, y_0, z_0)) = 0 \rightarrow$$

$$x^n - V_{y_0} - n \cdot x^{n-1}(x - V_{x_0}) - 1 = 0 \quad (3)$$

$$x^n + V_{y_0} - n \cdot x^{n-1}(x - V_{x_0}) - 1 = 0$$

$$n = 350, -1 \leq x \leq 1$$

Where the visibility boundary is the solution for these coupled equations. As can be noted, these equations are not related to Z axis, and the visibility boundary points are the same ones for each x-y surface due to the model's characteristics. Later on, we address the relations between a building's roof and visibility height in our visibility algorithm, as part of the visibility computation.

The visibility statement leads to two polynomial N order equations, which appear to be a complex computational task. The real roots of these polynomial equations are the solution to the visibility boundary. These equations can be solved efficiently by finding where the polynomial equation changes its sign and cross zero value; generating the real roots in a very short time computation (these functions are available in Matlab, Maple and other mathematical programs languages). Based on the polynomial cross zero solution, we can compute a fast and exact analytic solution for the visibility problem from a viewpoint to a 3D building model. This solution allows us to easily define the Visible Boundary Points.

Visible Boundary Points (VBP) - we define VBP of the object i as a set of boundary points $j=1..N_{bound}$ of the visible surfaces of the object, from viewpoint $V(x_0, y_0, z_0)$.

$$VBP_{i=1}^{j=1..N_{bound}}(x_0, y_0, z_0) = \begin{bmatrix} x_1, y_1, z_1 \\ x_2, y_2, z_2 \\ \dots \\ x_{N_{bound}}, y_{N_{bound}}, z_{N_{bound}} \end{bmatrix} \quad (4)$$

Roof Visibility – The analytic solution in equation (3) does not treat the roof visibility of a building. We simply check if viewpoint height V_{z_0} is lower or higher than the building height $h_{max_{c_i}}$ and use this to decide if the roof is visible or not:

$$V_{z_0} \geq Z = h_{max_{c_i}} \quad (5)$$

If the roof is visible, roof surface boundary points are added to VBP. Roof visibility is an integral part of VBP computation for each building. Currently, we assume flat roof surfaces that will be extended to more complex roof models in our future work.

Two simple cases using the analytic solution from a visibility point to a building, including visible roofs, can be seen in Figure 6. The visibility point is marked in black, the visible parts colored in red, and the invisible parts colored in blue. The visible volumes are computed immediately with a

very low computation effort, without scanning all the model's points, as is necessary in LOS-based methods for such a case.

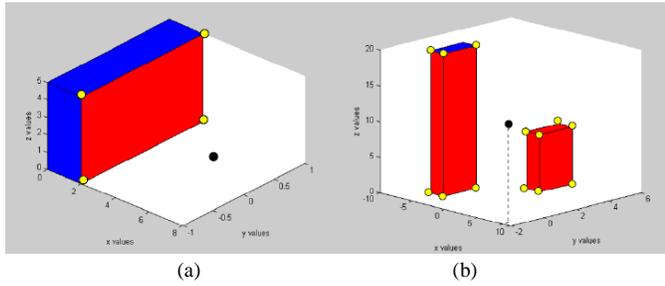


Figure 6. Visibility Volume computed with the Analytic Solution. Viewpoint is marked in black, visible parts colored in red and invisible parts in blue. VBP marked with yellow circles - (a) single building; (b) two non-overlapping buildings

B. Visibility Computation in Urban Environments

In the previous sections, we treated a single building case, without considering hidden surfaces between buildings, i.e. building surface occluded by other buildings, which directly affect the visibility volumes solution. In this section, we introduce our concept for dealing with these spatial relations between buildings, based on our ability to rapidly compute visibility volume for a single building generating VBP set.

Hidden surfaces between buildings are simply computed based on intersections of the visible volumes for each object. The visible volumes are easily defined using VBP, and are defined, in our case, as Visible Pyramids. The invisible components of the far building are computed by intersecting the projection of the closer buildings' VP base to the far building's VP base.

1) *The Visible Pyramid (VP)*: we define $VP_i^{j=1..N_{surf}}(x_0, y_0, z_0)$ of the object i as a 3D pyramid generated by connecting VBP of specific surface j to a viewpoint $V(x_0, y_0, z_0)$. Maximum number of N_{surf} for a single object is three. VP boundary, colored with green arrows, can be seen in Figure 7. The intersection of VPs allows us to efficiently compute the hidden surfaces in urban environments, as seen in the next sub-section.

2) *Hidden Surfaces between Buildings*: As we mentioned earlier, invisible parts of the far buildings are computed by intersecting the projection of the closer buildings' VP to the farther buildings' VP base.

Let VP_1^i, VP_2^j be visible pyramid from a viewpoint $V(x_0, y_0, z_0)$, The Projected Surface $PS_{VP_1^i}^{VP_2^j}$ from the closer buildings' VP_1^i to the farther buildings' VP_2^j base plane consists of projection of $VBP_1^{1..N_b}$ points:

$$PS_{VP_1^i}^{VP_2^j} = \begin{bmatrix} x_{p1}, y_{p1}, z_{p1} \\ x_{p2}, y_{p2}, z_{p2} \\ \dots \\ x_{pi}, y_{pi}, z_{pi} \\ \dots \\ x_{pN_{bound}}, y_{pN_{bound}}, z_{pN_{bound}} \end{bmatrix} \quad (6)$$

Where the normal of VP_2^j base plane is (a,b,c) and the plane can be written as $ax + by + cz + d = 0$. The projected point (x_{pi}, y_{pi}, z_{pi}) described in equation (6) is:

$$\begin{aligned} x_{pi} &= x_{VBP_i^j} - a \frac{ax_{VBP_i^j} + by_{VBP_i^j} + cz_{VBP_i^j} + d}{a^2 + b^2 + c^2} \\ y_{pi} &= y_{VBP_i^j} - b \frac{ax_{VBP_i^j} + by_{VBP_i^j} + cz_{VBP_i^j} + d}{a^2 + b^2 + c^2} \\ z_{pi} &= z_{VBP_i^j} - c \frac{ax_{VBP_i^j} + by_{VBP_i^j} + cz_{VBP_i^j} + d}{a^2 + b^2 + c^2} \end{aligned} \quad (7)$$

The Intersected Surface $IS_{VP_1^i}^{VP_2^j}$, between $PS_{VP_1^i}^{VP_2^j}$ and VP_2^j base plane can generally describe as polygons intersection:

$$IS_{VP_1^i}^{VP_2^j} = PS_{VP_1^i}^{VP_2^j} \cap \partial VP_2^j \cup \partial PS_{VP_1^i}^{VP_2^j} \cap VP_2^j \quad (8)$$

The Intersected Surface $IS_{VP_1^i}^{VP_2^j}$ is also the invisible one from a viewpoint $V(x_0, y_0, z_0)$, as can be seen in Figure 9.

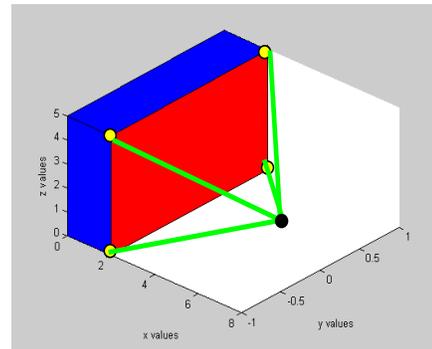


Figure 7. A Visible Pyramid from a viewpoint (marked as a black dot) to VBP of a specific surface

For simplicity, we demonstrate the method with two buildings from a viewpoint $V(x_0, y_0, z_0)$ one (denoted as the first one) of which hides, fully or partially, the other (the second one).

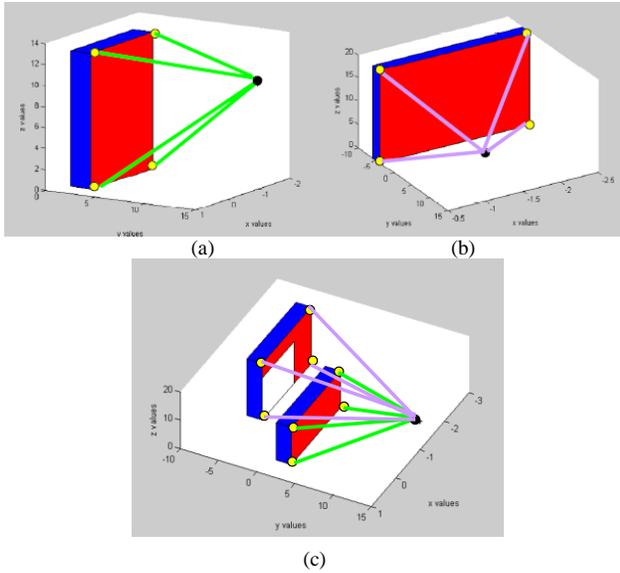


Figure 8. Generating VP - (a) VP_1^j boundary colored in green lines; (b) VP_2^j boundary colored in purple lines; (c) the two buildings - VP_1^j in green and VP_2^j in purple, and intersected surface in white

As seen in Figure 8, in this case, we first compute VBP for each building separately, $VBP_1^{1..4}$, $VBP_2^{1..4}$; based on these VBPs, we generate VPs for each building, VP_1^j , VP_2^j . After that, we project VP_1^j base to VP_2^j base plane, as seen in Figure 9, if existing. At this point, we intersect the projected surface in VP_2^j base plane, $PS_{VP_1^j}^{VP_2^j}$, and update $VBP_2^{1..4}$ and VP_2^j (removing the intersected part). The intersected part is the invisible part of the second building from viewpoint $V(x_0, y_0, z_0)$ which is hidden by the first building is $IS_{VP_1^j}^{VP_2^j}$ (marked in white in Figure 9).

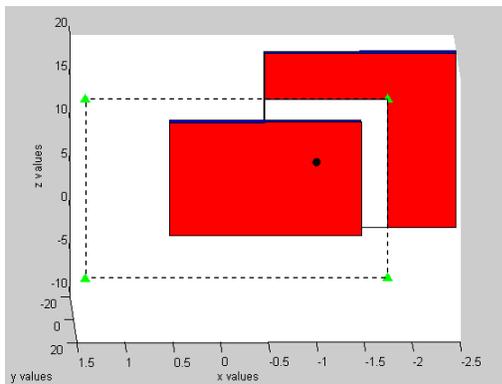


Figure 9. Projection of VP_1^j to VP_2^j base plane (projected surface) marked by dotted lines

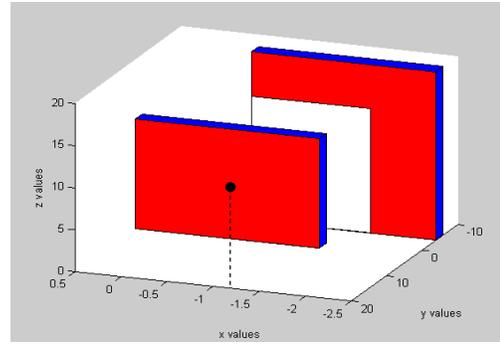


Figure 10. Computing Hidden Surfaces between Buildings by using the Intersected surface on VP_2^j base Plane.

In a case of a third building, in addition to the buildings presented in Figure 10, the projected VP will only be the visible ones, and the VBP and VP of the second building will be updated accordingly (as described in the next subsection). In cases of several buildings, the VP base would not necessarily be rectangular, due to the intersected surface profile of previous projections. We demonstrated a simple case of an occluded building. A general algorithm for a more complex scenario, which contains the same actions between all the combinations of VP between the objects, is detailed in the next sub-section. Projection and intersection of 3D pyramids can be done with simple computational geometry elements, which demand a very low computation effort.

C. Visibility Analysis for a Basic Shape Vocabulary

In this section, we present an analysis of visibility aspects of a basic shape vocabulary, as part of the mass modeling of urban environments. Mass modeling shapes consist of boxes as a basic structure, in different shapes such as *L*, *T*, *U*, and *H*. Based on visibility analysis for a single box, and the hidden surfaces removal between overlapping boxes introduced above, we demonstrate an accurate and fast visibility solution for mass modeling buildings profiles.

1) *L Shape Visibility*: We demonstrate visibility analysis for an L shape, which can be split into two separate boxes. The profile shape consists of boxes which overlap the visible surfaces, in some cases of the viewpoint location. Let the L shape be separated into two boxes A (Figure 11(a)) and B (Figure 11(b)), visible parts are colored in green, and invisible parts are colored in purple. We compute the VBP of each box - VBP_A , VBP_B . In the next phase, a visible pyramid is computed for each box - VP_A^1 , VP_B^1 . Projection of VP_A^1 to VP_B^1 base plane and intersection between pyramids are colored in black in these figures. The final visible part of L shape can be seen in Figure 11(c). A similar case, with a different viewpoint regarding the L shape, can be seen in Figure 12.

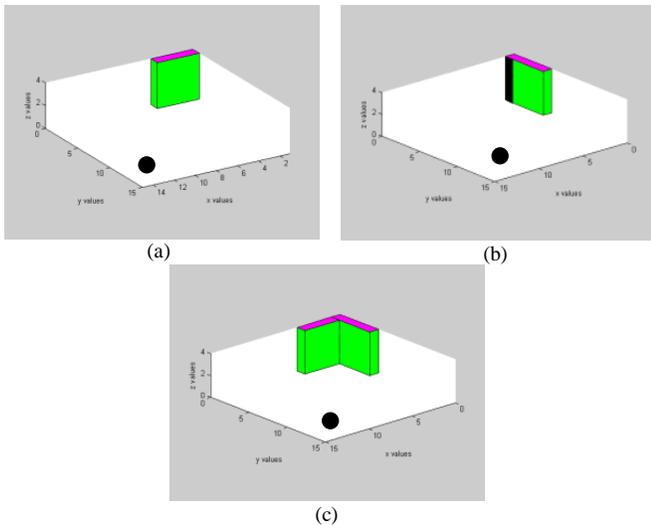


Figure 11. *L* Shape Visibility Analysis - (a) Box A and Viewpoint; (b) Box B and Viewpoint where the Hidden Surface Removal is colored in black; (c) *L* Shape with the visible and invisible parts

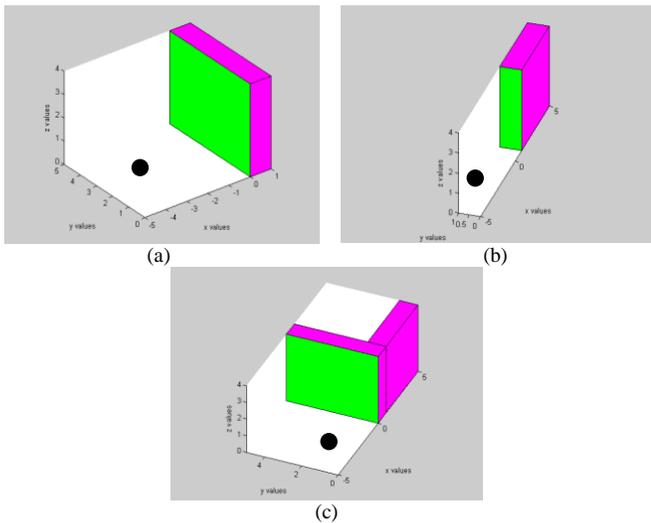


Figure 12. *L* Shape Visibility Analysis - (a) Box A and Viewpoint; (b) Box B and Viewpoint; (c) *L* Shape with the aggregated visible and invisible parts

2) *T Shape Visibility*: In this case, we demonstrate visibility analysis for a T shape, which can be split into two separate boxes (similar to the L shape case) – A (Figure 13(a)) and B (Figure 13(b)), the visible parts are colored in green and invisible parts in purple; and the viewpoint V colored by a black dot. We compute the VBP of each box - VBP_A, VBP_B . In the next phase, a visible pyramid is computed for each box - VP_A^1, VP_B^1 . Projection of VP_A^1 to VP_B^1 base plane and intersection between pyramids (colored with black) can be seen in Figure 13(c). The final visible part of the T shape can be seen in Figure 13(d).

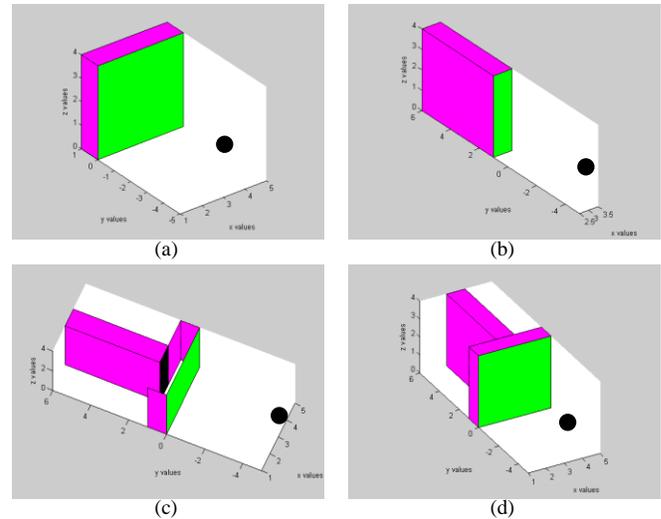
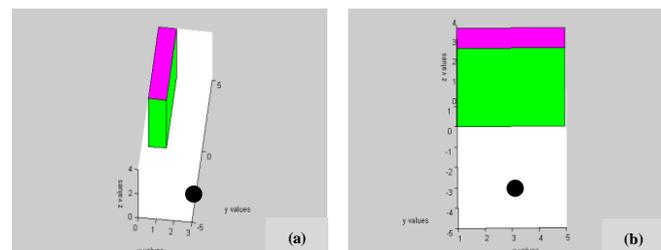


Figure 13. *T* Shape Visibility Analysis - (a) Box A and Viewpoint; (b) Box B and Viewpoint; (c) Hidden Surface Removal colored in black and visible surface colored in green; (d) *T* Shape with the visible and invisible parts

3) *U Shape Visibility*: In this case, we demonstrate the visibility analysis for a U shape, separated into three different boxes - A (Figure 14(a)), B (Figure 14(b)) and C (Figure 14(c)), visible parts are colored in green and invisible parts in purple; and the viewpoint V colored by a black dot. We compute the VBP of each box. In the next phase, a visible pyramid is computed for each box. The outcome of the projection and intersection between visible pyramids can be seen in Figure 14(d) and 14(e), colored with black. The final visible parts of the U shape can be seen in Figure 14(f).

4) *H Shape Visibility*: In this case, we demonstrate visibility analysis for an H shape, separated into three different boxes – A (Figure 15(a)), B (Figure 15(b)) and C (Figure 15(c)), visible parts are colored in green and invisible parts in purple; and the viewpoint V colored by a black dot. We compute the VBP of each box. In the next phase, a visible pyramid is computed for each box. The outcome of the projection and intersection between visible pyramids are colored in black. The final visible part of the H shape can be seen in Figure 15(d) and 15(e) from two different views.



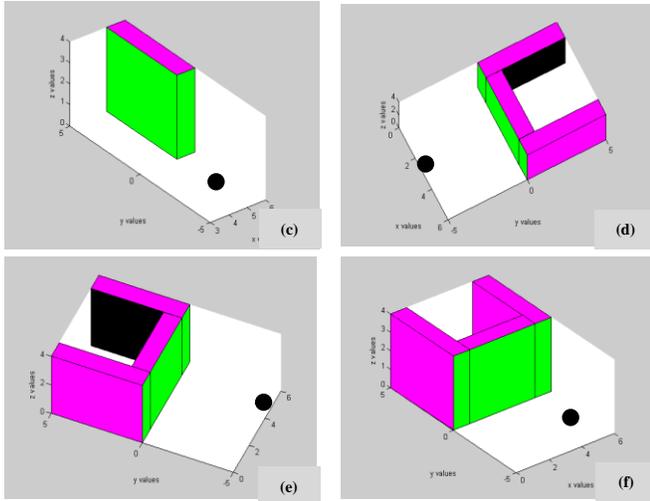


Figure 14. U Shape Visibility Analysis - (a) Box A and Viewpoint with visible part colored in green; (b) Box B and Viewpoint with visible part colored in green; (c) Box C and Viewpoint with visible part colored in green; (d) - (e) Hidden Surface Removal colored in black and visible surface colored in green; (f) U Shape with the visible and invisible parts

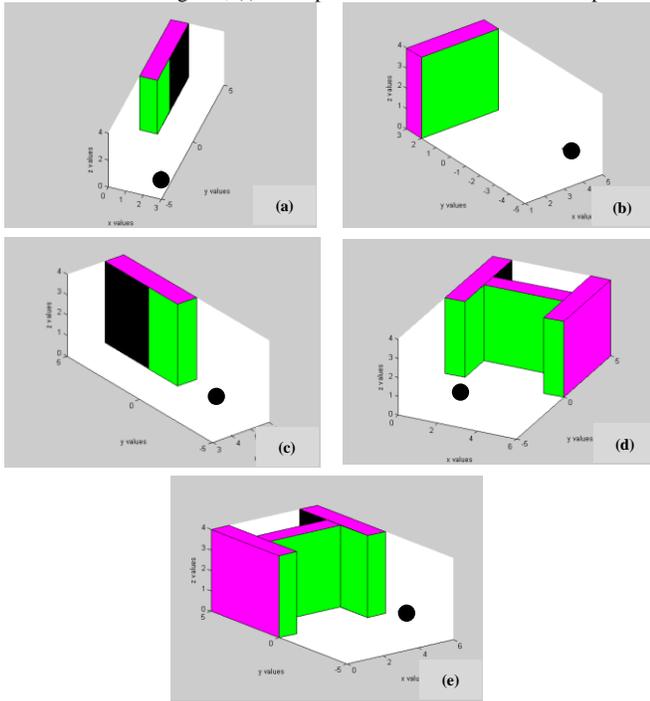


Figure 15. H Shape Visibility Analysis - (a) Box A and Viewpoint with visible part colored in green; (b) Box B with visible part colored in green; (c) Box C with visible part colored in green and Hidden Surface Removal colored in black ; (d)-(e) H Shape with the visible and invisible parts

D. Visibility Algorithm Pseudo - Code

1. Given viewpoint $V(x_0, y_0, z_0)$
2. For $i=1:1:N_{models}$ building model
 - 2.1. Calculate Azimuth θ_i and Distance D_i from viewpoint to object
 - 2.2. Set and Sort Buildings Azimuth Array $\theta[i]$
 - 2.3. IF Azimuth Objects ($i, i..i-1$) Intersect

2.3.1. Sort Intersected Objects $j=1:1:N_{intersect}$
By Distance

2.3.2. Compute VBP for each intersected building, $VBP_{j=1..N_{intersect}}^{1..N_{bound}}$

2.3.3. Generate VP for each intersected building, $VP_{j=1..N_{intersect}}^{1..N_{surf}}$

2.3.4. For $j=1:1:N_{intersect}-1$

2.3.4.1. Project $VP_j^{1..N_{surf}}$ base to

$VP_{j+1}^{1..N_{surf}}$ base plane, if exist.

2.3.4.2. Intersect projected surfaces in $VP_{j+1}^{1..N_{surf}}$ base plane.

2.3.4.3. Update $VBP_{j+1}^{1..N_{bound}}$ and $VP_{j+1}^{1..N_{surf}}$

End

Else

Locate Building in Urban Environment

End

End

E. Visibility Algorithm – Complexity Analysis

We analyze our algorithm complexity based on the pseudo code presented in the previous section, where n represents the number of buildings. In the worst case, n buildings hide each other. Visibility complexity consists of generating VBP and VP for n buildings, $nO(1)$ complexity. Projection and intersection are also $nO(1)$ complexity.

The complexity of our algorithm, without considering data structure managing for urban environments, is $nO(n)$.

1. $O(1)$
2. $O(n)$
- 2.1. $O(1)$
- 2.2. $O(1)$ – Data structure operator
- 2.3. $O(1)$ – Data structure operator
 - 2.3.1. $O(1)$ – Data structure operator
 - 2.3.2. $n \cdot O(1)$
 - 2.3.3. $n \cdot O(1)$
 - 2.3.4. $O(1)$ – Data structure operator
 - 2.3.4.1. $n \cdot O(1)$
 - 2.3.4.2. $n \cdot O(1)$
 - 2.3.4.3. $n \cdot O(1)$

We analyze the visibility algorithm complexity of the LOS methods, where n represents the number of buildings and k represents the resolution of the object. The exact visibility computation requires scanning each object and each object's points, $O(nk)$ where usually $k \gg n$.

VI. RESULTS

We have implemented the presented algorithm and tested some urban environments on a 1.8GHz Intel Core CPU with Matlab. First, we analyze the versatility of our algorithm on four different test scenes with different

occluded elements. These test scenes can be seen in Figures (17)-(20).

After that, we compare our algorithm to the basic LOS visibility computation, to prove accuracy and computational efficiency.

Urban environments modeled with mass modeling of a built-up environment consisting of basic shape vocabulary were also tested. First, we analyzed the versatility of our algorithm on a synthetic test scene with different occluded elements (Figure 19) and then on real data -Gibson House Museum Region, Beacon St, MA, USA (Figure 20).

A. Computation Time and Comparison to LOS

The main contribution of this research focuses on a fast and accurate visibility computation in urban environments. We compare our algorithm time computation with the common LOS visibility computation demonstrating our algorithm's computational efficiency.

1) *Visibility Computation Using LOS*: The common LOS visibility methods require scanning all of the object's points. For each point, we check if there is a line connecting the viewpoint to that point which does not cross other objects. We used the LOS2 Matlab function, which computes the mutual visibility between two points on a Digital Elevation Model (DEM) model. We converted our last test scene (Figure 20) with one to 58 buildings to DEM, operated LOS2 function, and measured CPU time after model conversion. Each building with DEM was modeled homogeneously by 50 points. The visible parts using the LOS method were the exact parts computed by our algorithm. The computation time of the LOS method was about 10,130 times longer than that of our analytic solution in this scene (4,257 sec vs. 0.42 sec). The CPU times of our analytic solution and the LOS method are depicted in Figure 16.

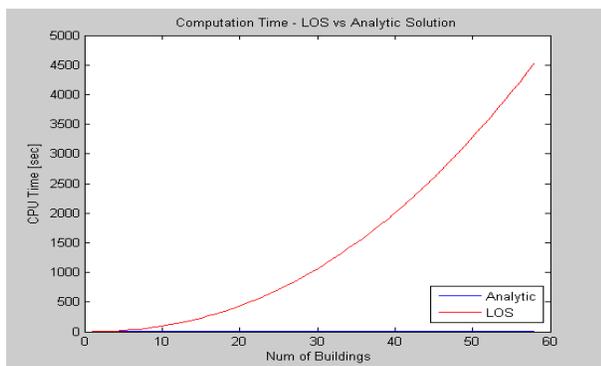


Figure 16. CPU Computation Times of the LOS and our algorithm. CPU was measured with an increasing number of buildings from one to 58. LOS method took 10,130 times longer than our algorithm

In case of mass modeling (Figure 19), computation time of the LOS method was about 6,600 times longer than that of our analytic solution (1,650 sec vs. 0.25 sec).

Over the last years, efficient LOS-based visibility methods for DEM models, such as Xdraw, have been

introduced in order to generate approximate solutions [7]. However, the computation time of these methods is at least $O(n(n-1))$, and, above all, the solution is only an approximate one.

VII. CONCLUSIONS AND FUTURE WORK

We have presented an efficient algorithm for visibility computation in an urban environment, modeling basic building structure with mathematical approximating for presentation of buildings' corners. Our algorithm is based on a fast visibility boundary computation for a single object, and on computing the hidden surfaces between buildings by using projected surfaces and intersections of the visible pyramids.

We have presented an extension from a basic box to a complex urban environment model using the basic shapes vocabulary of mass modeling. Each shape of this modeling can be sub-divided into several boxes, which stand for a basic building structure.

The main contribution of the method presented in this paper is that it does not require special hardware, and is suitable for on-line computations based on the algorithms' performances, as presented above. The method generates an exact and quick solution to the visibility problem in relatively complex urban environments, modeled or generated by using procedural modeling consisting of basic shape vocabulary, which can be used for real urban environments, as seen in Scene no. 3. Using these basic shapes, one can create buildings having different shapes (including, for example, balconies).

Complexity analysis of our algorithm has been presented, as well as the computational running time compared to the LOS visibility computation showing significant improvement of time performance.

Further research will focus on facing multi-viewpoints for optimal visibility computation in such environments, generalizing the presented building model such as cylinders and cones taking into account Level of Details (LOD) and roof modeling.

REFERENCES

- [1] O. Gal and Y. Doytsher, "Fast and Accurate Visibility Computation in a 3D Urban Environment", in Proc. of the Fourth International Conference on Advanced Geographic Information Systems, Applications, and Services, Valencia, 2012, pp: 105-110.
- [2] G. Drettakis and E. Fiume, "A Fast Shadow Algorithm for Area Light Sources Using Backprojection," In Computer Graphics (Proceedings of SIGGRAPH '94), 1994, pages 223-230.
- [3] G. Elber, R. Sayegh, G. Barequet, and R. Martin, "Two-Dimensional Visibility Charts for Continuous Curves," Shape Modeling International 05, MIT, Boston, USA, 2005, pp. 206-215.
- [4] Y. Chrysanthou, "Shadow Computation for 3D Interactive and Animation," Ph.D. Dissertation, Department of Computer Science, College University of London, UK, 1996.
- [5] H. Plantinga and R. Dyer, "Visibility, Occlusion, and Aspect Graph," The International Journal of Computer Vision, vol. 5(2), pp.137-160, 1990.

- [6] Y. Doytsher and B. Shmutter, "Digital Elevation Model of Dead Ground," Symposium on Mapping and Geographic Information Systems (Commission IV of the International Society for Photogrammetry and Remote Sensing), Athens, Georgia, USA, 1994.
- [7] W.R. Franklin and C. Ray, "Higher isn't Necessarily Better: Visibility Algorithms and Experiments," In T. C. Waugh & R. G. Healey (Eds.), *Advances in GIS Research: Sixth International Symposium on Spatial Data Handling*, 1994, pp. 751–770. Taylor & Francis, Edinburgh.
- [8] G. Nagy, "Terrain Visibility," Technical report, Computational Geometry Lab, ECSE Dept., Rensselaer Polytechnic Institute, 1994
- [9] W.R. Franklin, "Siting Observers on Terrain," in D. Richardson and P. van Oosterom, eds, *Advances in Spatial Data Handling: 10th International Symposium on Spatial Data Handling*. Springer-Verlag, 2002, pp. 109–120
- [10] W.R. Franklin and C. Vogt, "Multiple Observer Siting on Terrain with Intervisibility or Lores Data," in XXth Congress, International Society for Photogrammetry and Remote Sensing, Istanbul, 2004.
- [11] J. Wang, G.J. Robinson, and K. White, "A Fast Solution to Local Viewshed Computation Using Grid-based Digital Elevation Models," *Photogrammetric Engineering & Remote Sensing*, vol. 62, pp.1157-1164, 1996.
- [12] D. Cohen-Or and A. Shaked, "Visibility and Dead- Zones in Digital Terrain Maps," *Eurographics*, vol. 14(3), pp. 171- 180, 1995.
- [13] J. Wang, G.J. Robinson, and K. White, "Generating Viewsheds without Using Sightlines," *Photogrammetric Engineering & Remote Sensing*, vol. 66, pp. 87-90, 2000.
- [14] C. Ratti, "The Lineage of Line: Space Syntax Parameters from the Analysis of Urban DEMs'," *Environment and Planning B: Planning and Design*, vol. 32, pp. 547-566, 2005.
- [15] D. Fisher-Gewirtzman and I.A. Wagner, "Spatial Openness as a Practical Metric for Evaluating Built-up Environments," *Environment and Planning B: Planning and Design* vol. 30(1), pp. 37-49, 2003.
- [16] P.P.J. Yang, S.Y. Putra, and W. Li, "Viewsphere: a GIS-based 3D Visibility Analysis for Urban Design Evaluation," *Environment and Planning B: Planning and Design*, vol. 43, pp.971-992, 2007.
- [17] L. De Floriani and P. Magillo, "Visibility Algorithms on Triangulated Terrain Models," *International Journal of Geographic Information Systems*, vol. 8(1), pp. 13-41, 1994.
- [18] L. De Floriani and P. Magillo, "Intervisibility on Terrains," In P.A. Longley, M.F. Goodchild, D.J. Maguire & D.W. Rhind (Eds.), *Geographic Information Systems: Principles, Techniques, Management and Applications*, 1999, pp. 543-556. John Wiley & Sons.
- [19] B. Nadler, G. Fibich, S. Lev-Yehudi, and D. Cohen-Or, "A Qualitative and Quantitative Visibility Analysis in Urban Scenes," *Computers & Graphics*, 1999, pp. 655-666.
- [20] S. J. Teller, "Computing the Antipenumbra of an Area Light Source," *Computer Graphics*, vol. 26(2), pp.139-148, 1992.
- [21] J. Stewart and S. Ghali, "Fast Computation of Shadow Boundaries Using Spatial Coherence and Backprojections," In *Computer Graphics, Proceedings of SIGGRAPH 1994*, pp. 231-238.
- [22] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario, "Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes," In *EUROGRAPHICS'98*, 1998.
- [23] G. Schaufler, J. Dorsey, X. Decoret, and F.X. Sillion, "Conservative Volumetric Visibility with Occluder Fusion," In *Computer Graphics, Proceedings of SIGGRAPH 2000*, pp. 229-238.
- [24] F. Dowling and U. Flemming, "The bungalows of buffalo" *Environment and Planning B* 8, 1981, 269–293.
- [25] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, "Instant architecture". *ACM Transactions on Graphics* 22, 3, 2003, 669–677.
- [26] P. Muller, P. Wonka, S. Hawgler, A. Ulmer, and L.C. Gool, "Procedural Modeling of Buildings" In *Proceedings of ACM, SIGGRAPH 2006*, pp. 614-623.
- [27] G. Schmitt, *Architectura et machina*. 1993, Vieweg&Sohn.
- [28] S. Zlatanova, A. Rahman, and S. Wenzhong, "Topology for 3D Spatial Objects," *International Symposium and Exhibition on Geoinformation*, 2002, pp. 22-24.

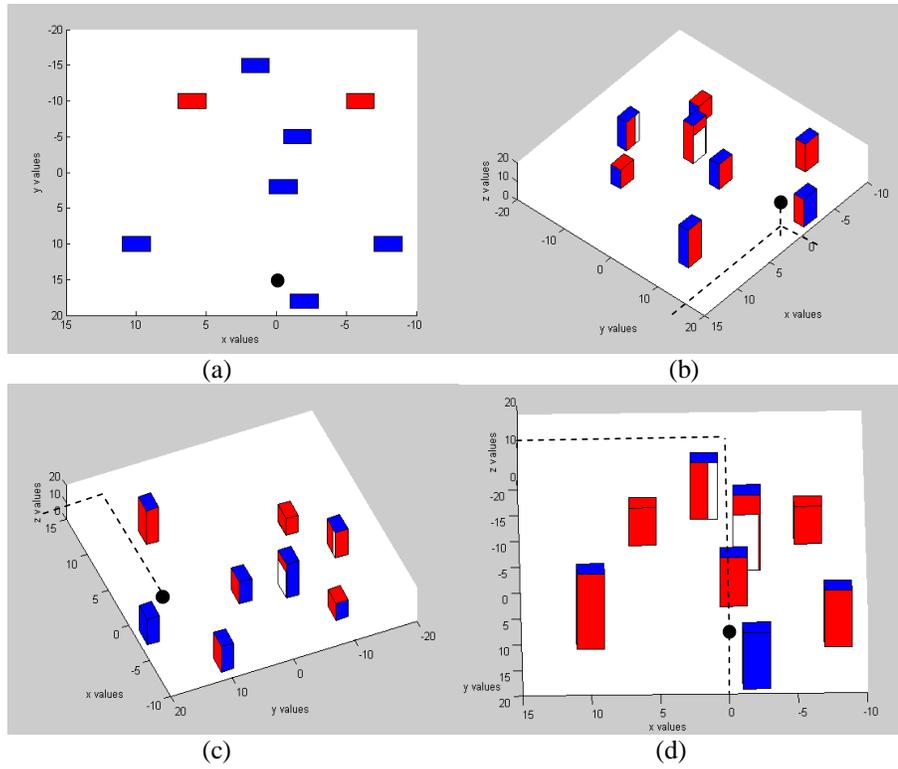


Figure 17. Scene number 1 - Eight buildings in an Urban Environment, $V(x_0, y_0, z_0) = (0, 15, 10)$ - (a) Topside view; (b)-(d) Different views demonstrating the visibility computation using our algorithm. CPU time was 0.15 sec

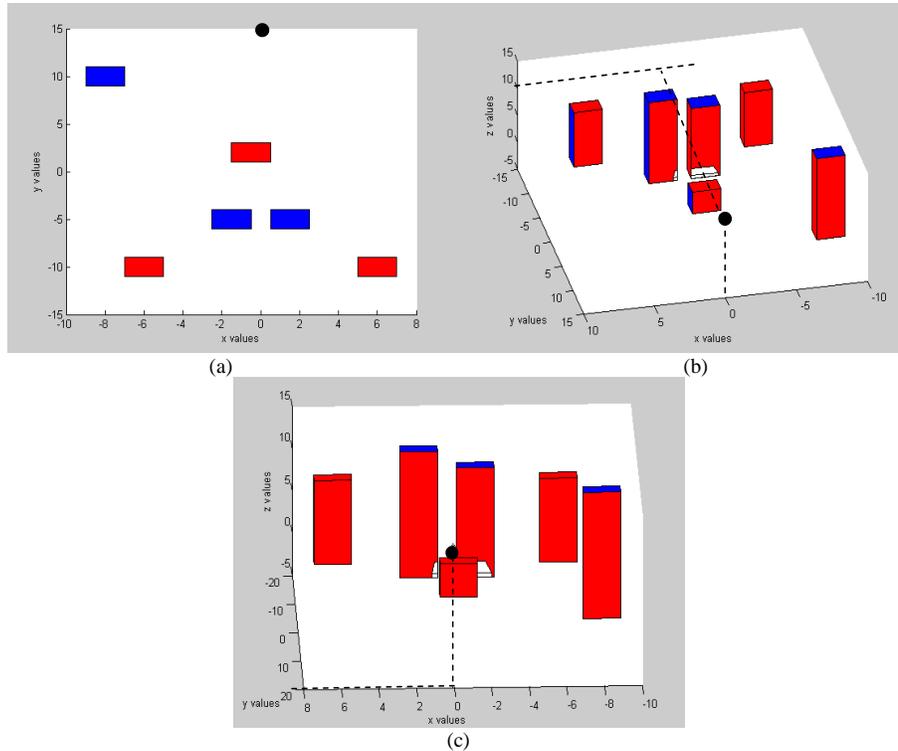


Figure 18. Scene number 2 - Six Buildings in an Urban Environment, where viewpoint is higher than the projected building, $V(x_0, y_0, z_0) = (0, 15, 10)$ - (a) Topside view; (b)-(c) Different views demonstrating visibility computation using our algorithm. CPU time was 0.14 sec

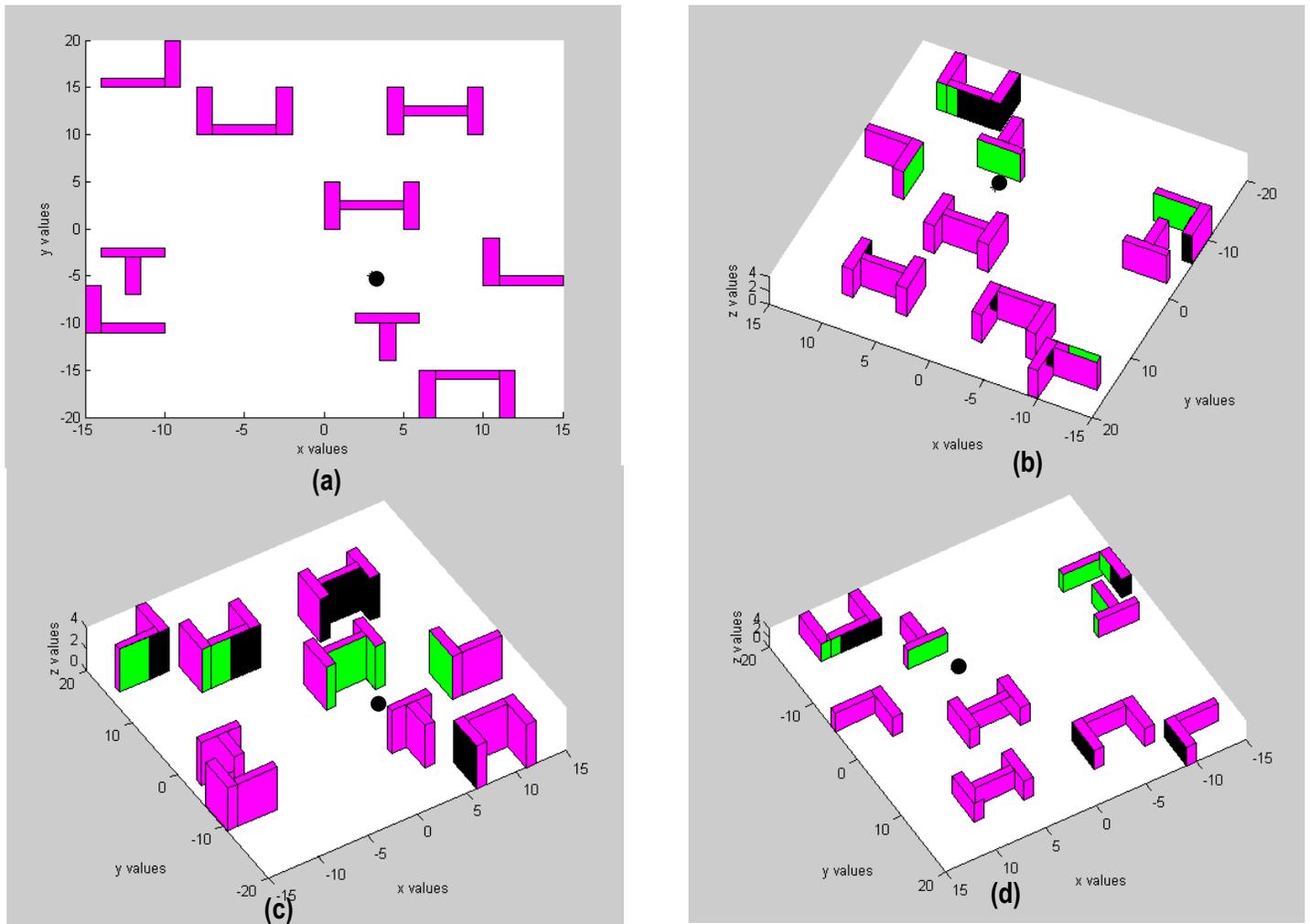


Figure 19. Scene number 3 - Nine basic shape structures of buildings in an Urban Environment, $V(x_0, y_0, z_0) = (3, -5, 2)$ - (a) Topside view; (b)-(d) Different views demonstrating the visibility computation using our algorithm. CPU time was 0.25 sec

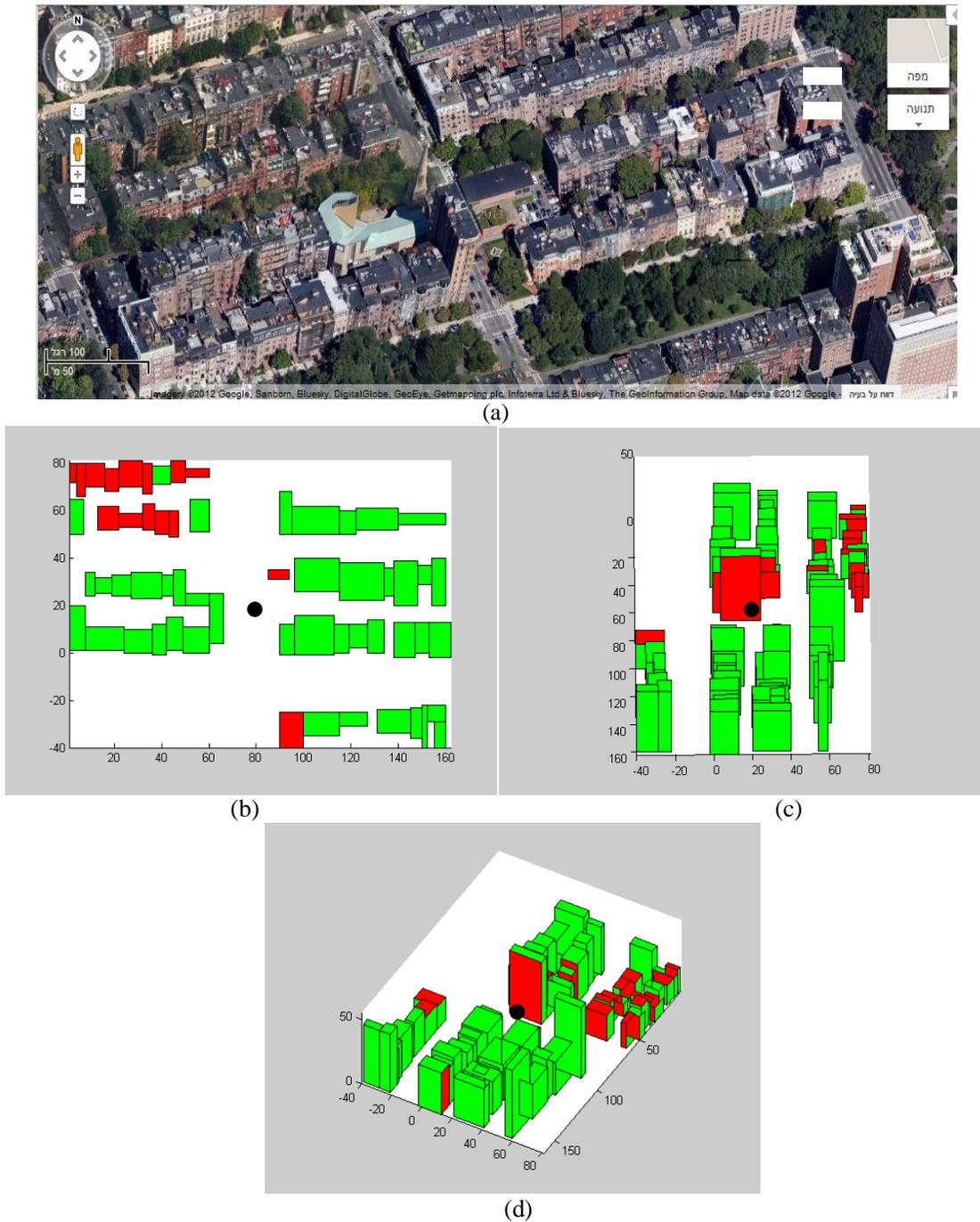


Figure 20. (a) Scene number 4 - Real Data of Urban Environments. (a) Gibson House Museum Region, Beacon St, MA, USA (Google Maps); Visible parts colored in red and invisible parts with green (b) Topview Modeling; (c)-(d) Sideviews. $V(x_0, y_0, z_0) = (80, 20, 20)$. CPU time was 0.42 sec



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, COCOR, PESARO, INNOV

✦ issn: 1942-2601