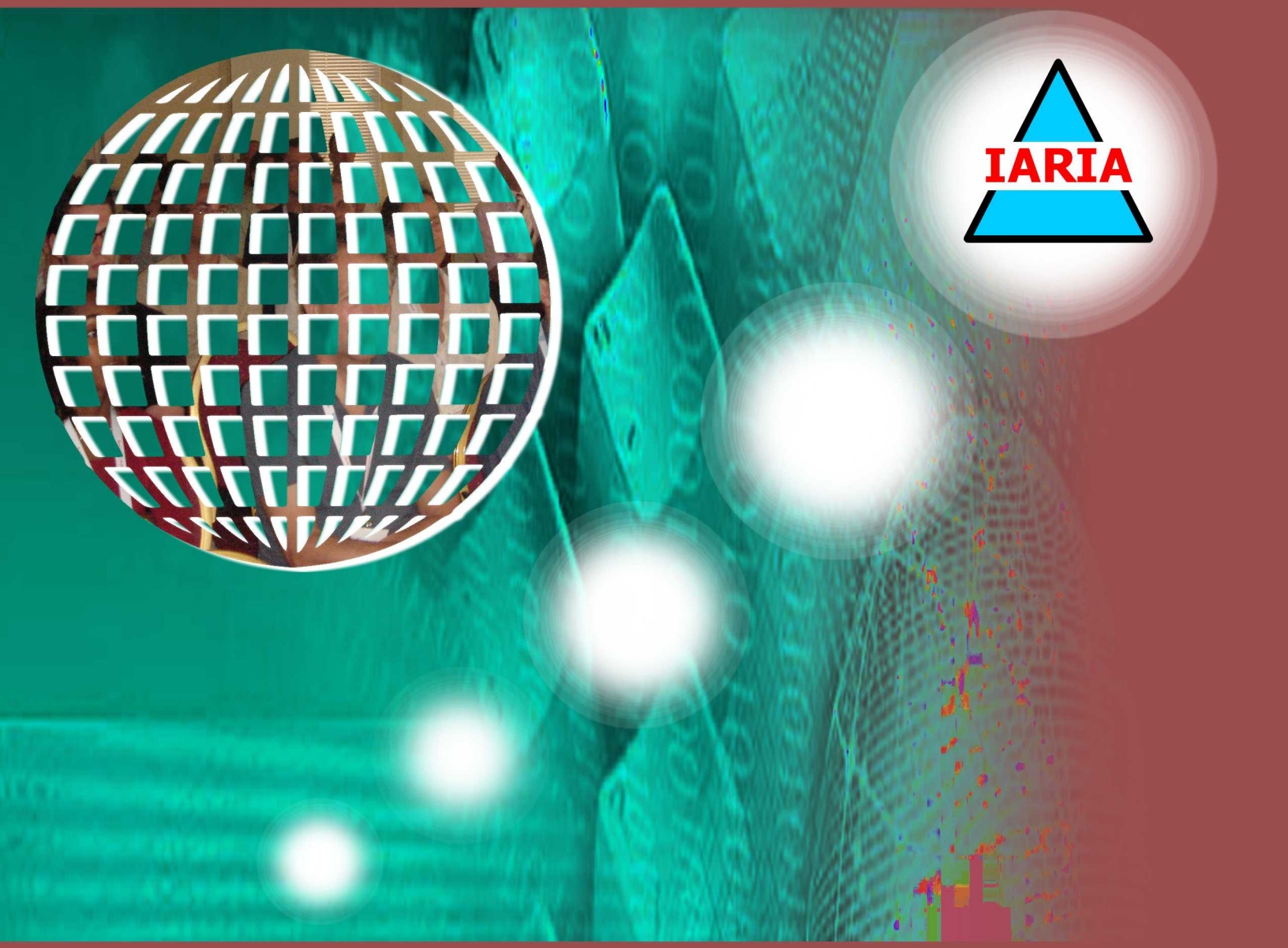
International Journal on Advances in Software



2020 vol. 13 nr. 1&2

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

Sponsored by IARIA
www.iaria.org

International Journal on Advances in Software
Volume 13, Number 1 & 2, 2020

Muneera Bano, International Islamic University - Islamabad, Pakistan
Fabian Barbato, Technology University ORT, Montevideo, Uruguay
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
Jorge Bernardino, ISEC - Institute Polytechnic of Coimbra, Portugal
Rudolf Berrendorf, Bonn-Rhein-Sieg University of Applied Sciences - Sankt Augustin, Germany
Ateet Bhalla, Independent Consultant, India
Fernando Boronat Seguí, Universidad Politecnica de Valencia, Spain
Pierre Borne, Ecole Centrale de Lille, France
Farid Bourennani, University of Ontario Institute of Technology (UOIT), Canada
Narhimene Boustia, Saad Dahlab University - Blida, Algeria
Hongyu Pei Breivold, ABB Corporate Research, Sweden
Carsten Brockmann, Universität Potsdam, Germany
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 Ana G. Méndez, 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
Yaser Chaaban, Leibniz University of Hanover, Germany
Savvas A. Chatzichristofis, Democritus University of Thrace, Greece
Antonin Chazalet, Orange, France
Jiann-Liang Chen, National Dong Hwa University, China
Shiping Chen, CSIRO ICT Centre, Australia
Wen-Shiung Chen, National Chi Nan University, Taiwan
Zhe Chen, College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China PR
Po-Hsun Cheng, National Kaohsiung Normal University, Taiwan
Yoonsik Cheon, The University of Texas at El Paso, USA
Lau Cheuk Lung, INE/UFSC, Brazil
Robert Chew, Lien Centre for Social Innovation, Singapore
Andrew Connor, Auckland University of Technology, New Zealand
Rebeca Cortázar, University of Deusto, Spain
Noël Crespi, Institut Telecom, Telecom SudParis, France
Carlos E. Cuesta, Rey Juan Carlos University, Spain
Duilio Curcio, University of Calabria, Italy
Mirela Danubianu, "Stefan cel Mare" University of Suceava, Romania
Paulo Asterio de Castro Guerra, Tapijara Programação de Sistemas Ltda. - Lambari, Brazil
Cláudio de Souza Baptista, University of Campina Grande, Brazil
Maria del Pilar Angeles, Universidad Nacional Autonónoma de México, México
Rafael del Vado Vírseda, Universidad Complutense de Madrid, Spain
Giovanni Denaro, University of Milano-Bicocca, Italy
Nirmit Desai, IBM Research, India
Vincenzo Deufemia, Università di Salerno, Italy
Leandro Dias da Silva, Universidade Federal de Alagoas, Brazil
Javier Diaz, Rutgers University, USA
Nicholas John Dingle, University of Manchester, UK
Roland Dodd, CQUniversity, Australia
Aijuan Dong, Hood College, USA

Suzana Dragicevic, Simon Fraser University- Burnaby, Canada
Cédric du Mouza, CNAM, France
Ann Dunkin, Palo Alto Unified School District, USA
Jana Dvorakova, Comenius University, Slovakia
Hans-Dieter Ehrich, Technische Universität Braunschweig, Germany
Jorge Ejarque, Barcelona Supercomputing Center, Spain
Atilla Elçi, Aksaray University, Turkey
Khaled El-Fakih, American University of Sharjah, UAE
Gledson Elias, Federal University of Paraíba, Brazil
Sameh Elnikety, Microsoft Research, USA
Fausto Fasano, University of Molise, Italy
Michael Felderer, University of Innsbruck, Austria
João M. Fernandes, Universidade de Minho, Portugal
Luis Fernandez-Sanz, University of de Alcala, Spain
Felipe Ferraz, C.E.S.A.R, Brazil
Adina Magda Florea, University "Politehnica" of Bucharest, Romania
Wolfgang Fohl, Hamburg Universiy, Germany
Simon Fong, University of Macau, Macau SAR
Gianluca Franchino, Scuola Superiore Sant'Anna, Pisa, Italy
Naoki Fukuta, Shizuoka University, Japan
Martin Gaedke, Chemnitz University of Technology, Germany
Félix J. García Clemente, University of Murcia, Spain
José García-Fanjul, University of Oviedo, Spain
Felipe Garcia-Sanchez, Universidad Politecnica de Cartagena (UPCT), Spain
Michael Gebhart, Gebhart Quality Analysis (QA) 82, Germany
Tejas R. Gandhi, Virtua Health-Marlton, USA
Andrea Giachetti, Università degli Studi di Verona, Italy
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, AristaFlow GmbH, Germany
Carlos Granell, European Commission / Joint Research Centre, Italy
Christoph Grimm, University of Kaiserslautern, Austria
Michael Grottke, University of Erlangen-Nuernberg, Germany
Vic Grout, Glyndwr University, UK
Ensar Gul, Marmara University, Turkey
Richard Gunstone, Bournemouth University, UK
Zhensheng Guo, Siemens AG, Germany
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
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
Alla Lake, Linfo Systems, LLC, USA
Fritz Laux, Reutlingen University, Germany
Luigi Lavazza, Università dell'Insubria, Italy
Fábio Luiz Leite Júnior, Universidade Estadual da Paraiba,Brazil
Alain Lelu, University of Franche-Comté / LORIA, France
Cynthia Y. Lester, Georgia Perimeter College, USA
Clement Leung, Hong Kong Baptist University, Hong Kong
Weidong Li, University of Connecticut, USA
Corrado Loglisci, University of Bari, Italy
Francesco Longo, University of Calabria, Italy
Sérgio F. Lopes, University of Minho, Portugal
Pericles Loucopoulos, Loughborough University, UK
Alen Lovrencic, University of Zagreb, Croatia
Qifeng Lu, MacroSys, LLC, USA
Xun Luo, Qualcomm Inc., USA
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
Gerasimos Marketos, University of Piraeus, Greece
Abel Marrero, Bombardier Transportation, Germany
Adriana Martin, Universidad Nacional de la Patagonia Austral / Universidad Nacional del Comahue, Argentina
Goran Martinovic, J.J. Strossmayer University of Osijek, Croatia
Paulo Martins, University of Trás-os-Montes e Alto Douro (UTAD), Portugal
Stephan Mäs, Technical University of Dresden, Germany
Constandinos Mavromoustakis, University of Nicosia, Cyprus
Jose Merseguer, Universidad de Zaragoza, Spain
Seyedeh Leili Mirtaheri, Iran University of Science & Technology, Iran

Lars Moench, University of Hagen, Germany
Yasuhiko Morimoto, Hiroshima University, Japan
Antonio Navarro Martín, Universidad Complutense de Madrid, Spain
Filippo Neri, University of Naples, Italy
Muaz A. Niazi, Bahria University, Islamabad, Pakistan
Natalja Nikitina, KTH Royal Institute of Technology, Sweden
Roy Oberhauser, Aalen University, Germany
Pablo Oliveira Antonino, Fraunhofer IESE, Germany
Rocco Oliveto, University of Molise, Italy
Sascha Opletal, Universität Stuttgart, Germany
Flavio Oquendo, European University of Brittany/IRISA-UBS, France
Claus Pahl, Dublin City University, Ireland
Marcos Palacios, University of Oviedo, Spain
Constantin Paleologu, University Politehnica of Bucharest, Romania
Kai Pan, UNC Charlotte, USA
Yiannis Papadopoulos, University of Hull, UK
Andreas Papasalouros, University of the Aegean, Greece
Rodrigo Paredes, Universidad de Talca, Chile
Päivi Parviainen, VTT Technical Research Centre, Finland
João Pascoal Faria, Faculty of Engineering of University of Porto / INESC TEC, Portugal
Fabrizio Pastore, University of Milano - Bicocca, Italy
Kunal Patel, Ingenuity Systems, USA
Óscar Pereira, Instituto de Telecomunicacoes - University of Aveiro, Portugal
Willy Picard, Poznań University of Economics, Poland
Jose R. Pires Manso, University of Beira Interior, Portugal
Sören Pirk, Universität Konstanz, Germany
Meikel Poess, Oracle Corporation, USA
Thomas E. Potok, Oak Ridge National Laboratory, USA
Christian Prehofer, Fraunhofer-Einrichtung für Systeme der Kommunikationstechnik ESK, Germany
Ela Pustułka-Hunt, Bundesamt für Statistik, Neuchâtel, Switzerland
Mengyu Qiao, South Dakota School of Mines and Technology, USA
Kornelije Rabuzin, University of Zagreb, Croatia
J. Javier Rainer Granados, Universidad Politécnica de Madrid, Spain
Muthu Ramachandran, Leeds Metropolitan University, UK
Thurasamy Ramayah, Universiti Sains Malaysia, Malaysia
Prakash Ranganathan, University of North Dakota, USA
José Raúl Romero, University of Córdoba, Spain
Henrique Rebêlo, Federal University of Pernambuco, Brazil
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
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
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
Daniel Schall, Vienna University of Technology, Austria
Rainer Schmidt, Munich University of Applied Sciences, Germany

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
Cristian Stanciu, University Politehnica of Bucharest, Romania
Lena Strömbäck, SMHI, Sweden
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
Božo Tomas, HT Mostar, Bosnia and Herzegovina
Davide Tosi, Università degli Studi dell'Insubria, Italy
Guglielmo Trentin, National Research Council, Italy
Dragos Truscan, Åbo Akademi University, Finland
Chrisa Tsinaraki, Technical University of Crete, Greece
Roland Ukor, FirstLinq Limited, UK
Torsten Ullrich, Fraunhofer Austria Research GmbH, Austria
José Valente de Oliveira, Universidade do Algarve, Portugal
Dieter Van Nuffel, University of Antwerp, Belgium
Shirshu Varma, Indian Institute of Information Technology, Allahabad, India
Konstantina Vassilopoulou, Harokopio University of Athens, Greece
Miroslav Velev, Aries Design Automation, USA
Tanja E. J. Vos, Universidad Politécnica de Valencia, Spain
Krzysztof Walczak, Poznan University of Economics, Poland
Yandong Wang, Wuhan University, China
Rainer Weinreich, Johannes Kepler University Linz, Austria
Stefan Wesarg, Fraunhofer IGD, Germany
Wojciech Wiza, Poznan University of Economics, Poland
Martin Wojtczyk, Technische Universität München, Germany
Hao Wu, School of Information Science and Engineering, Yunnan University, China
Mudasser F. Wyne, National University, USA
Zhengchuan Xu, Fudan University, P.R.China
Yiping Yao, National University of Defense Technology, Changsha, Hunan, China
Stoyan Yordanov Garbatov, Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento, INESC-ID, Portugal
Weihai Yu, University of Tromsø, Norway
Wenbing Zhao, Cleveland State University, USA
Hong Zhu, Oxford Brookes University, UK
Qiang Zhu, The University of Michigan - Dearborn, USA

International Journal on Advances in Software
Volume 13, Numbers 1 & 2, 2020

CONTENTS

Jan-Hendrik Martenson, NetApp Deutschland GmbH, Germany
Jannes Gless, Center for Information Management, Hannover Medical School, Germany
Branko Jandric, Center for Information Management, Hannover Medical School, Germany
Joshua Görner, Airbus Operations GmbH, Germany
Detlef Amendt, Center for Information Management, Hannover Medical School, Germany
Benjamin Schantze, NetApp Deutschland GmbH, Germany
Svetlana Gerbel, Center for Information Management, Hannover Medical School, Germany

An Explanation Framework for Whole Processes of Data Analysis Applications: Concepts and Use Cases

Hiroshi Ishikawa
Graduate School of Systems Design
Faculty of System Design
Tokyo Metropolitan University
Hino, Tokyo
Email:ishikawa-hiroshi@tmu.ac.jp

Yukio Yamamoto
Japan Aerospace Exploration Agency
Sagamihara, Kanagawa
Email:yamamoto.yukio@jaxa.jp

Masaharu Hirota
Department of Information Science
Faculty of Informatics
Okayama University of Science
Okayama, Okayama
Email:hirota@mis.ous.ac.jp

Masaki Endo
Division of Core Manufacturing
Polytechnic University
Kodaira, Tokyo
Email:endou@uitec.ac.jp

Abstract- **The main contribution of the paper is to address the necessity of both macro and micro explanations for Social Big Data (SBD) applications and to propose an explanation framework integrating both, allowing SBD applications to be more widely accepted and used. The framework provides both a macro explanation of the whole procedure and a micro explanation of the constructed model and an explanation of the decisions made by the model. Application systems including Artificial Intelligence (AI) or Data Mining (DM) need reproducibility to ensure their reliability as scientific systems. For that purpose, it is important to illustrate the procedures of the system explicitly and abstractly (that is, macro explanations). This paper has scientific value in that it proposes a data model for that purpose and illustrates the possibility of macro explanations through one use case of social science. Scientists also need to provide evidence that the results obtained by AI or DM are valid. In other words, this paper also has scientific value in that it reveals how the features of the model and concrete grounds for judgment can be explained through two use cases of natural science.**

Keywords- social big data; explanayion; data model; data management; data mining.

I. INTRODUCTION

We are surrounded by big data, which are waiting to be analyzed and used. Big data are real data, such as automobile driving data and space observation data, generated from real world measurement and observation, social data derived from social media, e.g., Twitter and Instagram, and open data published by highly public groups, e.g., weather data and evacuation location data. These are generally called social big data (SBD) [1]. Furthermore, SBD are inherently represented by multimedia (MM). By integrating and analyzing social big data, new knowledge can be obtained, which is expected to bring new value to society [2] [3].

SBD can include the same type of spatially different data, data obtained by different means for the same object, and temporally different data for the same object as well. Therefore, SBD applications cover not only use cases that include social data, but also use cases that include only engineering or scientific data generated in the real world.

Further as the horizon of applications whose main task is data analysis spreads, the following problems have emerged:

- ***Application to science, e.g., lunar and planetary science***

Analytical applications in this field require strictness as science. That is, explanation of the protocol (procedure) of analysis and explanation of the reason for decisions are required [1]. In addition, as to the interpretation of the analytical model, it is necessary to explain the input data (for learning and test) and the data manipulation on the data, and the procedure (algorithm and program) for model construction. In order to interpret the individual results, it is necessary to explain the input data (actual data) and the reasons for the decisions.

- ***Application to Social Infrastructure, e.g., Mobility as a Service (MaaS)***

Analytical applications in this field require consent of practitioners. That is, the analysis result must be consistent with the practitioners' own experiences, and especially in the case of applications such as ones related to human life, it is

necessary to fulfill the accountability to the concerned parties. Interpretation of both of the model and individual results is necessary as with science. In addition, especially if the data about the generic users are utilized in applications, interpretation of the model is also important in order to get rid of the general users' concerns.

In order for social big data to widely be used, it is necessary to explain the user the application system. Both microscopic description, that is, interpretation of the analytical model and explanation of individual decisions and macroscopic description, that is, description of the whole process including the data manipulation and the model construction are required.

First of all, the reason why a macro explanation is necessary is described below. In order for social big data applications to be accepted by users, it is necessary to ensure at least their reliability. Since information science is one area of science, we should guarantee reproducibility as science. In other words, it is necessary to ensure that third parties can prepare and analyze data according to given explanation and can get the same results.

In addition, in order for the service to be operatable, it is necessary for the final user of the service to be convinced of how the service processes and uses the personal information. In addition, if the users can be convinced of the description of way of using the personal information, the progress of data portability can be advanced based on the EU's GDPR (General Data Protection Regulation) law on personal information protection [4] and Japan-based information bank to promote the use of personal information [5].

Next, a micro explanation is necessary for the following reasons. In order for analysts of social big data and field experts using the data to accept decisions made by the constructed model, it is assumed that they must understand the structure, actions and grounds of the model and are satisfied with them as well.

Up to now, the authors have been involved in the development of a wide range of social big data use cases ranging from tourism, disaster prevention to lunar and planetary science [6] [7]. In the course of these processes, from the users of the use cases, we have often received questions as to what kind of data are processed, what kind of models are created as the core of analysis, and furthermore, what are the grounds for the decisions. In other words, from the development experiences of multiple use cases, we have come to think that both the macro explanation proposed in this paper and the micro explanation emerging in AI are urgently needed.

To date, the authors created multiple seismic source classifiers of the lunar quakes (i.e., moonquakes) in the field of lunar and planetary science using the Balanced Random Forest [8], and the features, e.g., the distance between the moon and the earth, were calculated and studied for extracting features strongly related to the cause of moonquakes as a micro explanation [6]. With regard to a macro explanation, the authors also showed that by observing many use cases, social big data applications should include different digital ecosystems such as data management (database operation) and data analysis (data mining, machine learning, artificial intelligence), we have noticed that it is necessary to have a method to generally describe the whole process of application consisting of such a hybrid digital ecosystem. Therefore, as a framework to describe processes in an abstraction level independent of a specific programming language, we have come to think of adopting a data model [9] developed in the field of database and proposed a framework for its description using the mathematical concept of family of sets [10]. As described in the subsequent section of the related works, the research on micro explanations is being actively carried out, whereas as far as research on the framework for a macroscopic description is not known except for our work.

The main contribution of the paper is to address the necessity of both macro and micro explanations for SBD applications and to propose an explanation framework integrating both of them. This will allow SBD applications to be more widely accepted and used. Although this paper describes our research-in-progress, we propose an integrated framework for explanation and introduce a part of its functions through case studies.

The contributions of this paper can be detailed as follows. First, as a science, a system that includes Artificial Intelligence (AI) or Data Mining (DM) needs reproducibility (How) [11] to ensure its reliability. For that purpose, it is important to show the procedures of the system explicitly and abstractly. We propose a dedicated data model for that purpose. For AI or DM, scientists need to show what model features are useful for making decisions and why the results obtained are valid (Why) [12]. This paper has scientific value in clarifying what features contribute more to the classification model and what can be shown as a basis for individual judgment through two use cases. The procedure can be modeled using a data model approach based on the mathematical concept *family* [10], using social data in the first case related to social science. The difference method [13] was used in the first case and the third case, related to natural science in order to model the hypotheses. In the explanation of the features of the analytical model in the second case, there is a skew in the data size for moonquake data, so we used Balanced Random Forest [8]. In the third case, for the basis of individual judgment we used CNN (deep learning) [14] and Grad-CAM (attention) [15] using Digital Elevation Model (DEM) provided by the Japan Aerospace Exploration Agency (JAXA).

This paper is of scientific value in that it demonstrates through use cases what can be explained to scientists as a basis for validating the results obtained by AI or DM. In other words, moonquake classification is important in lunar and

planetary science to understand the internal structure of the moon. This paper illustrated which features contribute most to the classification. The crater with a central hill is also a promising place for exploring the internal structure of the moon. This paper could illustrate what is the basis for judging the craters with central hills. These are also scientifically significant in that they have shown the possibility that AI and DM, which are IT technologies, are accepted by scientists as scientific methods.

The differences between this paper and the international conference paper [1] are as follows. The contribution and scientific value of this study were described in more detail. A description of the basic elements of the framework for explanation and the mechanism of its processing was added. A case of discovery of lunar craters with central hills was added as an example of a microscopic-explanation function (i.e., description of the grounds of judgment). The description of each use case was summarized according to the items such as scientific objectives, data, methods, and results.

In Section II, we will introduce our explanation framework. Through use case examples of macroscopic description and microscopic description, we will describe features of the proposed approach in Sections III, IV, and V, respectively.

II. OUR APPROACH

A. Explanation Framework

For a macro explanation of applications, the goal is to facilitate a data model for abstractly describing the entire processes from data acquisition to data analysis and to explain the processes based on the description. For the micro explanation, we aim to show the basis of the interpretation of the constructed model and the individual decisions made when applying it.

Macroscopic-explanation function (F1)
= data management procedure + model generation procedure
Microscopic-explanation function
= model feature explanation (F2) + judgement basis explanation (F3)

Figure 1. Explanation framework

The features of the proposed framework are summarized as follows.

Based on the SBD model introduced in Section III, the parties who are users of the framework (application developers) can describe the procedures (i.e., data management and model generation) of the application system in a more abstract manner than programming languages. The framework outputs the described procedures as they are as a macro explanation to the parties (e.g., tourism operators in the tourism case). As a micro explanation, based on the results of the actual execution of the classification model, the framework outputs the features of the classification model (i.e., which features contribute to the classification) and the basis for judgment of each classification result to the scientists in the moonquake case and those in the moon crater case as the parties, respectively.

Figure 1 shows the framework. We specify which function corresponds to each use case. Case One in Section III illustrates the macroscopic-explanation function (F1) that explains the application procedures. Case Two in Section IV illustrates the microscopic-explanation function (F2) that explains which features contribute to the model classification, Case Three in Section V illustrates the microscopic-explanation function (F3) that explains the basis for individual judgment of the model classification.

We describe the framework in more detail as follows.

1) Construction of a theoretical foundation for integrated explanation

For that purpose, we build a theoretical framework of the technical foundation that integrates the following microscopic- and macroscopic-explanatory methods.

a) Macro explanation function: The application system is a hybrid ecosystem consisting of data management and data mining (including machine learning and Artificial Intelligence, or AI), and the function must be able to describe the application seamlessly. Moreover, it must be able to describe the application in a high level not depending on individual environments or programming languages. For instance, we aim to enable to describe “partition foreign visitors’ tweets into grids based on geo-tags.” Therefore, we first create a framework to unify the hybrid ecosystem based on the data model approach. In other words, we develop a method to provide macro explanations with the constituent elements (data structure and data manipulation) of the model based on the mathematical family of sets as a basic unit. The explanation mechanism provided by the proposed framework presents as a macro explanation a sequence of operations on databases to the user based on the model of SBD applications consisting of data management and data mining as in a use case depicted in Section III.

b) Microscopic-explanatory function: We develop an explanatory method independent of analytical model by extending explanatory functions based on attributes or constituent elements, which is an emergent approach in AI, discussed in the related work subsection. In other words, in model categories for structured data consisting of attributes, such as Support Vector Machine (SVM) and decision trees, we develop a method for systematically discovering subsets of attributes with strong influence on analysis results based on multiple weak classifiers. For instance, we aim to enable to illustrate a possibility that the features of the Earth and some of the features of Jupiter are effective for classification of the moonquakes when the moon is the origin of the

coordinate system. Especially this function is used to interpret the model itself. In model categories like Deep Neural Network (DNN) suitable for non-structured data such as images, we develop a method of explaining the analysis result based on the constituent elements or decomposition of the image with the use of annotation or attention. Especially this function is used to show the basis of individual decisions. For the micro explanation of the reasons for decisions, if the analysis target is image data, a part of the image which leads to the conclusion is indicated by concepts or words as its annotations based on a heat map. For instance, we aim to enable to illustrate that the contribution area for "central-peak crater" on the moon has heating area inside the crater, and the heating area is covering a central peak. If the object is structural data, that is, it consists of attributes, the micro explanation is presented in terms of the contribution ratios of the attributes as in a use case depicted in Section IV.

Please also note that data management and model construction in SBD applications are more complex than linear model construction frequent in traditional applications.

2) Collection of use cases and verification of basic technology

First, we collect several different kinds of use cases (tourism, mobility service, lunar exploration). We generate concrete explanations as targets for typical ones, using the integrated explanatory platform developed in items *a* and *b* and verify its feasibility

3) Implementation of Explanation generation and presentation method

Based on the theoretical framework of the integrated infrastructure, an automatic generation method of explanation and a presentation function of explanations are implemented. We evaluate their effectiveness by performing the experiments. We also incorporate InfoGraphics [16] as a method of presenting explanations to users since the users are not always analysis experts.

Basically, for micro explanation, we create explanations of individual decisions by solving partial problems that restrict information existing in original problems.

In this research, we aim to develop both the emerging microscopic-explanatory functions and macroscopic-explanatory functions and to build a framework for integrating two kinds of explanations.

B. Related Research

As a trend other than the authors' research, research corresponding to microscopic-explanatory functions has become active in AI, what is so called eXplainable AI (XAI) at present.

First, there is an attempt [17] to try to give a basic definition to the possibility of interpretation of a model in machine learning and research [18] on the evaluation method of interpretability.

Next, individual studies on XAI are roughly classified into (1) description based on features, (2) interpretable model, and (3) derivation of explanation model. Research is done to create a classification rule for explanation by creating a subset of features in SVM as a category of (1) [19]. In addition, in the image classification using Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM), there is research to generate explanations based on both image features and class features [20]. Further there is research introducing the explanation vector to make explicit the most important attributes [21]. In the category of (2), there is research using a AND/OR tree to discover the components of the model [22] and research to make models that can be interpreted by considering the generation process of features [23]. Research deriving description with reference of any classifier of the local approximation model falls into the category (3) [24].

In particular, the paper [25] is related to our framework. *Post-hoc global* explanation introduced in this paper corresponds to the explanation of the model features (micro explanation F2) in the proposed framework, *Post-hoc local* explanation introduced in the paper corresponds to the explanation of the basis for the judgement (micro explanation F3). However, the paper differs from our work in that the former takes no account of the macro explanation F1 (data management and model generation) in the proposed framework.

While developing along the approaches of (1) and (3) as a microscopic-explanatory technique, we aim to build a comprehensive explanation basis by conducting research on macroscopic-explanation technology.

In addition, although there is an application of infographics to a tourism use case [26], our research aims at basic research that can be widely used for visualization of explanation of general analysis.

III. CASE STUDY: MACRO EXPLANATION OF TOURISM APPLICATION

We will describe the case that explains how our data is used in analysis application. For that purpose, an integrated data model is introduced as a macroscopic description of an analytical application which is a hybrid ecosystem. Thus, the application is described using the integrated model just as a basis for macro explanation (see Figure 1). In other words, the data model introduced for model construction and reuse in the previous works [3] [13] is used for different purposes, i.e., the explanation functions for hypothesis generation.

A. Integrated Model

In the following subsections, we describe our data model approach to SBD, which consists of both of data structures and operations [9].

1) Data model for SBD

Our SBD model uses a mathematical concept of a *family*

[10], a collection of sets, as a basis for data structures. Family can be used as an apparatus for bridging the gaps between data management operations and data analysis operations.

Basically, our database is a *Family*. A Family is divided into *Indexed family* and *Non-Indexed family.* A Non-Indexed family is a collection of sets.

An Indexed family is defined as follows:

a) *{Set} is a Non-Indexed family with Set as its element.*

b) *{Set_i} is an Indexed family with Set_i as its i-th element. Here, i: Index is called indexing set and i is an element of Index.*

c) *Set is {<time space object>}.*

d) *Set_i is {<time space object>}$_i$. Here, object is an identifier to arbitrary identifiable user-provided data, e.g., record, object, and multimedia data appearing in social big data. Time and space are universal keys across multiple sources of social big data.*

e) *{Indexed $family_i$} is also an Indexed family with Indexed $family_i$ as its i-th element. In other words, Indexed family can constitute a hierarchy of sets.*

Please note that the following concepts are interchangeably used in this paper.

- *Singleton family ⇔ set*
- *Singleton set ⇔ element*

As described later in this section, we can often observe that SBD applications contain families as well as sets and they involve both data mining and data management. Please note that a family is also suitable for representing hierarchical structures inherent in time and locations as well as matrices and tensors associated with social big data.

If operations constructing a family out of a collection of sets and those deconstructing a family into a collection of sets are provided in addition to both family-dedicated and set-dedicated operations, SBD applications will be described in an integrated fashion by our proposed model.

2) SBD Operations.

SBD model constitutes an algebra with respect to Family as follows. SBD is consisted of Family data management operations and Family data mining operations. Further, Family data management operations are divided into Intra Family operations and Inter Family operations.

First, Intra Family Data Management Operations will be described as follows:

a) *Intra Indexed Intersect (i:Index Db p(i)) returns a singleton family (i.e., set) intersecting sets which satisfy the predicate p(i). Database Db is an indexed Family, which will not be mentioned hereafter.*

b) *Intra Indexed Union (i:Index Db p(i)) returns a singleton family union-ing sets which satisfy p(i).*

c) *Intra Indexed Difference (i:Index Db p(i)) returns a singleton family, that is, the first set (i.e., a set with smallest index) satisfying p(i) minus all the rest of sets satisfying p(i)*

d) *Indexed Select (i:Index Db p(i) sp(i)) returns an Indexed family with respect to i (preserved) where the element sets satisfy the predicate p(i) and the elements of the selected sets satisfy the selection predicate sp(i). As a special case of true as p(i), this operation returns the whole indexed family. In a special case of a singleton family, Indexed Select is reduced to Select (a relational operation).*

e) *Indexed Project (i:Index Db p(i) a(i)) returns an Indexed family where the element sets satisfy p(i) and the elements of the sets are projected according to a(i), attribute specification. This also extends also relational Project.*

f) *Intra Indexed cross product (i:Index Db p(i)) returns a singleton family obtained by product-ing sets which satisfy p(i). This is extension of Cartesian product, one of relational operators.*

g) *Intra Indexed Join (i:Index Db p(i) jp(i)) returns a singleton family obtained by joining sets which satisfy p(i) based on the join predicate jp(i). This is extension of Join (a relational operator).*

h) *Select-Index (i:Index Db p(i)) returns i:Index of set_i which satisfy p(i). As a special case of true as p(i), it returns all index.*

i) *Make-indexed family (Index Non-Indexed Family) returns an indexed Family. This operator requires order-compatibility, that is, that i corresponds to i-th set of Non-Indexed Family.*

j) *Partition (i:Index Db p(i)) returns an Indexed family. Partition makes an Indexed family out of a given set (i.e. singleton family either w/ or w/o index) by grouping elements with respect to p (i:Index). This is extension of "groupby" as a relational operator.*

k) *ApplyFunction (i:Index Db f(i)) applies f(i) to i-th set of DB, where f(i) takes a set as a whole and gives another set including a singleton set (i.e., Aggregate function). This returns an indexed family. f(i) can be defined by users.*

Here the operations a) to g) are extensions of corresponding relational operators.

Second, Inter Family Data Management Operations will be described as follows:

All are assumed to be Index-Compatible.

a) *Indexed Intersect (i:Index Db1 Db2 p(i)) union-compatible*

b) *Indexed Union (i:Index Db1 Db2 p(i)) union-compatible*

c) *Indexed Difference (i:Index Db1 Db2 p(i)) union-compatible*

d) *Indexed Join (i:Index Db1 Db2 p1(i) p2(i))*

e) *Indexed cross product (i:Index Db1 Db2 p(i))*

Finally, Family Data Mining Operations will be described as follows:

a) *Cluster (Family method similarity {par}) returns a Family as default, where Index is automatically produced. This is an unsupervised learner.*

b) *Make-classifier (i:Index set:Family learnMethod {par}) returns a classifier (Classify) with its accuracy. This is a supervised learner.*

c) *Classify (Index/class set) returns an indexed family with class as its index.*

d) *Make-frequent itemset (Db supportMin) returns an Indexed Family as frequent itemsets, which satisfy supportMin.*

e) *Make-association-rule (Db confidenceMin) creates association rules based on frequent itemsets Db, which satisfy confidenceMin. This is out of range of our algebra, too.*

Please note that the predicates and functions used in the above operations can be defined by the users in addition to the system-defined ones such as Count.

B. Tourist Applications

First, we will summarize this case as follows.

a) *(Social scientific and explanatory objectives) In this case related to social science, it is important for the EBPM (Evidence-Based Policy Making) [27] parties (tourist operators) to identify where there is a gap between social needs (many foreigners want to use the Internet) and the infrastructure to meet them (free Wi-Fi access spots for foreigners are available). The procedure to realize this consists of data management and model generation (data mining and difference method). In other words, it is necessary to explain to the EBPM parties how to draw the conclusions (results of gap detection).*

b) *(Data used for use case) Social media data Flickr [28] images and Twitter [29] articles were used. We collected 4.7 million Tweet articles (tweets) with geo tags by using the site provided API and selected 7,500 tweets posted by foreign visitors in Yokohama. We also collected 0.6 million Flickr images by using the site provided API and selected 2,100 images posted by foreign visitors in Yokohama.*

c) *(Methods used for use case) We used SQL to prepare social data and used a dedicated DM technique [30] to select only data posted by foreign visitors. We calculated the final result by using the difference method [3] [13] on separate results obtained from to the different data sources.*

d) *(Result) As a result of social science, we could identify the areas with the gaps between social needs and available infrastructures. The model-based explanation of the whole processes for obtaining the result was found useful by talks with tourism operators.*

Next, we will describe the case in more depth.

We describe a case study, finding candidate access spots for accessible Free Wi-Fi in Japan [31]. This case is classified as integrated analysis based on two kinds of social data.

This section describes our proposed method of detecting attractive tourist areas where users cannot connect to accessible Free Wi-Fi by using posts by foreign travelers on social media.

Our method uses differences in the characteristics of two types of social media:

Real-time: Immediate posts, e.g., Twitter

Batch-time: Data stored to devices for later posts, e.g., Flickr

Twitter users can only post tweets when they can connect devices to Wi-Fi or wired networks. Therefore, travelers can post tweets in areas with Free Wi-Fi for inbound tourism or when they have mobile communications. In other words, we can obtain only tweets with geo-tags posted by foreign travelers from such places. Therefore, areas where we can obtain huge numbers of tweets posted by foreign travelers are identified as places where they can connect to accessible Free Wi-Fi and /or that are attractive for them to sightsee.

Flickr users, on the other hand, take many photographs by using digital devices regardless of networks, but whether they can upload photographs on-site depends on the conditions of the network. As a result, almost all users can upload photographs after returning to their hotels or home countries. However, geo-tags annotated to photographs can indicate when they were taken. Therefore, although it is difficult to obtain detailed information (activities, destinations, or routes) on foreign travelers from Twitter, Flickr can be used to observe such information. In this study, we are based on our hypothesis of "A place that has a lot of Flickr posts, but few Twitter posts must have a critical lack of accessible Free Wi-Fi." We extracted areas that were tourist attractions for foreign travelers, but from which they could not connect to accessible Free Wi-Fi by using these characteristics of social media. What our method aims to find is places currently without accessible Free Wi-Fi.

Our method envisaged places that met the following two conditions as candidate access spots for accessible free Wi-Fi:

- *Spots where there was no accessible Free Wi-Fi*
- *Spots that many foreign visitors visited*

We use the number of photographs taken at locations to extract tourist spots. Many people might take photographs of subjects, such as landscapes based on their own interests. They might then upload those photographs to Flickr. These locations at which many photographs had been taken might also be interesting places for many other people to sightsee or visit. We have defined such places as tourist spots. We specifically examined the number of photographic locations to identify tourist spots to find locations where photographs had been taken by a lot of people. We mapped photographs that had a photographic location onto a two-dimensional grid based on the location at which a photograph had been taken to achieve this. Here, we created individual cells in a grid that was 30 square meters. Consequently, all cells in the grid that was obtained included photographs taken in a range. We then counted the number of users in each cell. We regarded cells with greater numbers of users than the threshold as tourist spots.

[Integrated Hypothesis] Based on different data generated form Twitter and Flickr, the following fragment as the macro explanation for hypothesis generation discovers attractive tourist spots for foreign visitors but without accessible free Wi-Fi currently (see Figure 2):

$DB_{t/visitor}$ ← Tweet DB of foreign visitors obtained by mining based on durations of their stays in Japan;

$DB_{f/visitor}$ ← Flickr photo DB of foreign visitors obtained by mining based on their habitations;

T ← Partition (*i:Index grid* $DB_{t/visitor}$ $p(i)$); This partitions foreign visitors' tweets into grids based on geo-tags; This operation returns an indexed family.

F ← Partition (*j:Index grid* $DB_{f/visitor}$ $p(j)$); This partitions foreign visitors' photos into grids based on geo-tags; This operation returns an indexed family.

Index1 ← Select-Index (*i:Index T Density*(*i*) >= *th1*); *Density* counts the number of foreign visitors per grid. *th1* is a threshold. This operation returns a singleton family.

Index2 ← Select-Index (*i:Index F Density*(*i*) >= *th2*); *Density* also counts the number of foreign visitors per grid. *th2* is a threshold. This operation returns a singleton family.

Index3 ← Difference (*Index2 Index1*); This operation returns a singleton family.

Plaese note that Partition and Select-Index are family data management operations while Difference is a relational (set) data management operation.

We collected more than 4.7 million data items with geo-tags from July 1, 2014 to February 28, 2015 in Japan. We detected tweets tweeted by foreign visitors by using the method proposed by Saeki et al. [30]. The number of tweets that was tweeted by foreign visitors was more than 4.7 million. The number of tweets that was tweeted by foreign visitors in the Yokohama area was more than 7,500. We collected more than 0.6 million photos with geo-tags from July 1, 2014 to February 28, 2015 in Japan. We detected photos that had been posted by foreign visitors to Yokohama by using our proposed method. Foreign visitors posted 2,132 photos. For example, grids indexed by *Index3* contain "Osanbashi Pier." Please note that the above description doesn't take unique users into consideration. The visual comparison of the same grids with unbalanced densities can help the decision makers to understand the proposal.

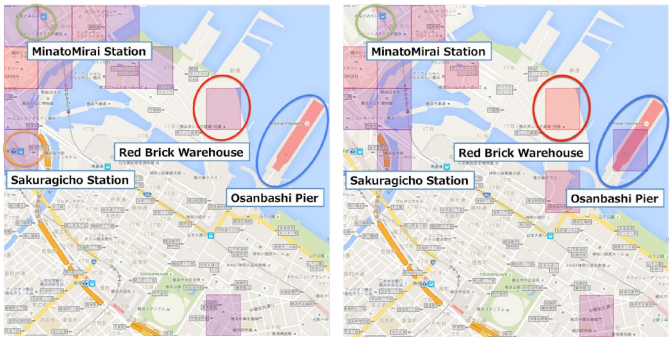


Figure 2. Differences of high-density areas of Tweets (left) and of Flickr photos (right).

IV. CASE STUDY: MICRO EXPLANATION FOR MOONQUAKE APPLICATION

First, we will summarize this case as follows.

a) *(Scientific and explanatory objectives) In this case related to lunar and planetary science, in order to know the internal structure of the moon, it is necessary to analyze the moonquake. As a preliminary study, the classification of moonquakes based on multiple epicenters is indispensable. However, it is not fully understood what features are more effective for moonquake classification. Therefore, it is necessary to determine the features that contribute most to the classification result as an explanation of the classification model.*

b) *(Data used for use case) We used passive seismic data regarding to the moonquakes collected by the NASA Apollo program. The dataset [32] has 16 seismic sources and 2,480 events as depicted in Table II. There is a skew with respect to the size of each source.*

c) *(Methods used for use case) We used Balanced Random Forest [8] to explain which features most contribute to one-to-one classification of moonquakes with respect to seismic sources.*

d) *(Result) Results of the classification performance using orbit parameters of objects in our Solar System (Earth, Sun, Jupiter, and Venus) suggest that the*

Earth orbit parameter is the most effective feature among them. The Jupiter orbit parameter is effective for classification of some seismic sources. The effect was validated by discussions in our research team consisting of IT specialist and natural scientists.

Next, we will describe the case of determining features important for interpreting the constructed model by reducing features with small contribution ratios. We apply Balanced Random Forest [8], which extends Random Forest [33], a popular supervised learning method in machine learning, to lunar and planetary science to verify the key features in analysis. Our verification method tries to confirm whether the known seismic source labels can be reproduced by Balanced Random Forest using the features described below based on the features constructed from the moonquakes with the seismic source label of the known moonquake as the correct label.

A. Features for Analysis

TABLE I shows the parameters in the coordinate systems used in this section. We use as seismic source of moonquakes the position on the planets of the moon, the sun, the earth, and Jupiter (X, y, z), velocity ($v x, v y, v z$), and distance (*lt*). Based on the time of moonquake occurrence, we calculate and use features using SPICE [34]. SPICE assists scientists in planning and interpreting scientific observations from space-borne instruments, and to assist engineers involved in modeling, planning and executing activities needed to conduct planetary exploration missions.

Here, sun perturbation is the solar perturbation. The IAU MOON coordinate system is a fixed coordinate system centered on the moon. The z axis is the north pole direction of the moon, the x axis is the meridian direction of the moon, the y axis is the right direction with respect to the plane xz. The IAU EARTH coordinate system is a fixed coordinate system centered on the earth. Here, the z axis is the direction of the conventional international origin, the x axis is the direction of the prime meridian, and the y axis is the right direction with respect to the xz plane.

We also calculate the period of the perigee at the distance of *earth_from_moon*, the period based on the period of the perigee, the periods of the *x* coordinate and the *y* coordinate of the solar perturbation. *sin* and *cos* values are calculated from these periodic features and the phase angle based on them. In addition, at the positions *moon_from_earth* and *sun_from_earth*, we calculate the *cos* similarity as the features of the sidereal moon. Most importantly, as all possible combinations of these features, a total of 55 features are used in our experiments described here.

B. Balanced Random Forest

Random Forest is an ensemble learning that combines a large number of decision trees and is widely used in fields such as data mining and has a characteristic that the contribution ratio of features can be calculated. However, Random Forest has a problem such that when there is a large difference in the size of data to be learned depending on class labels, the classifier is learned biased towards classes with a large size of data. Generally, we address the problem of imbalanced data by weighting classes with a small number of data. However, if there is any large skew between the numbers of data, the weight of data belonging to classes with a small number will become large, which is considered to cause over fitting to classes with a small number of data. Since the deep moonquakes have a large difference in the number of events for each seismic source, it is necessary to apply a method considering imbalanced data.

As analysis considering imbalanced data, we apply Balanced Random Forest [8], which makes the number of samples even for each class when constructing each decision tree. Balanced Random Forest divides each decision tree based on the Gini coefficient. Gini coefficient is an index representing impurity degree, which takes a value between 0 and 1. The closer it is to 0, the higher the purity is, that is, the less variance the data have. The contribution ratio of the feature is calculated for each feature by calculating the reduction ratio by the Gini coefficient at the branch of the tree. The final contribution ratio is the average value of contribution ratios of each decision tree.

C. Experiment Setting

Here, we describe experiments for evaluating features effective for seismic source classification, together with the results and considerations. Based on the classification performance and the contribution ratio of the features by Balanced Random Forest, we analyze the relationship between the seismic sources in the features used in this paper.

The outline of feature analysis is as follows: Features are calculated based on the time of occurrence of moonquake. Balanced Random Forest is applied to each pair of all seismic sources. Classification performance and the contribution ratio of the features by Balanced Random Forest are calculated and analyzed.

In this paper, as one-vs-one method, by constructing the classifier for every pair of two seismic sources in the dataset, we perform analysis paying attention to characteristics of each seismic source and the relationship between seismic sources. 100 Random Forests are constructed for each classifier. The number of samples used to construct each decision tree are taken 50 by bootstrap method. Bootstrap is a test or metric that relies on random sampling with replacement. Also, scikit-learn [35] was used to construct each decision tree in Random Forest. scikit-learn is a machine learning library for the Python programming language. In this paper, we perform the following analysis as feature selection.

• We create a classifier that learns all of the extracted 55 features.

TABLE I. PARAMETERS IN THE COORDINATE SYSTEMS COMPUTED USING SPICE.

Target	Observer	Coordinate system	Parameter
EARTH BARYCENTER	**MOON**	IAU MOON	**earth_from_moon**
SOLAR SYSTEM BARYCENTER	**MOON**	IAU MOON	**sun_from_moon**
JUPITER BARYCENTER	**MOON**	IAU MOON	**jupiter_from_moon**
SOLAR SYSTEM BARYCENTER	**EARTH BARYCENTER**	IAU EARTH	**sun_from_earth**
JUPITER BARYCENTER	**EARTHBARYCENTER**	IAU EARTH	**jupiter_from_earth**
SUN	**SOLAR SYSTEM BARYCENTER**	IAU EARTH	**sun_perturbation**

TABLE II. NUMBER OF DATA FOR EACH SEISMIC SOURCE.

Seismic source	A1	A5	A6	A7	A8	A9	A10	A14	A18	A20	A23	A25	A35	A44	A204	A218
Number of data	441	76	178	85	327	145	230	165	214	153	79	72	70	86	85	74

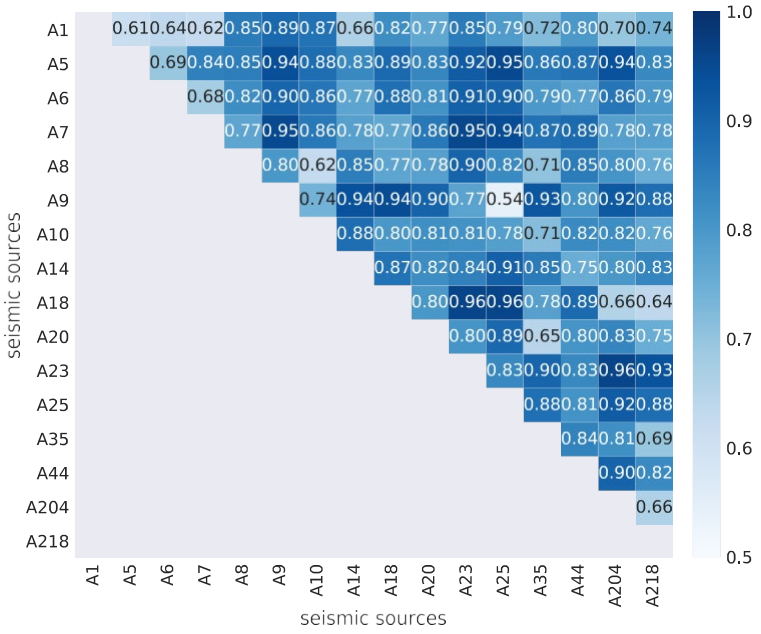


Figure 3. Averages of F-measures for pairs of seismic sources.

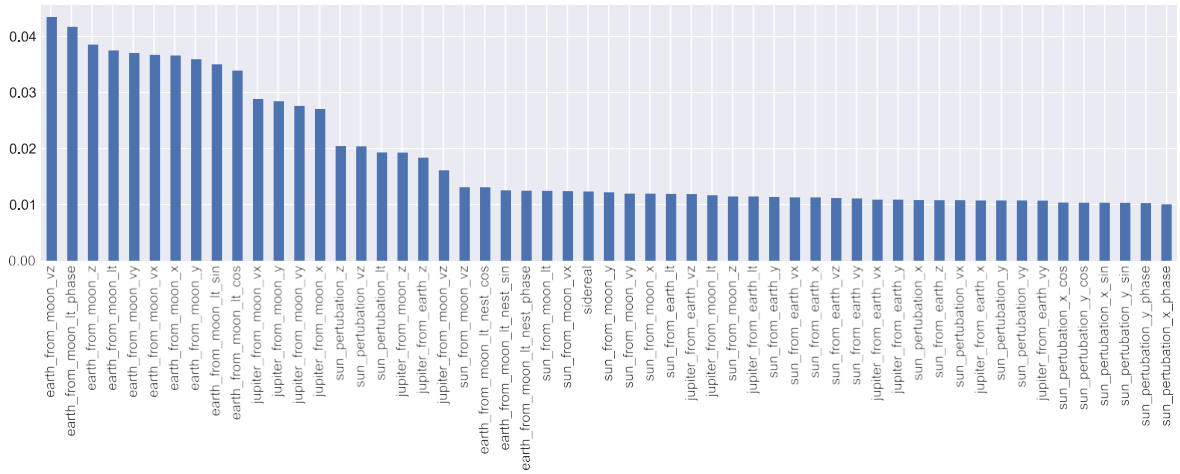


Figure 4. Averages of contribution ratios for each feature.

• Using the Variance Inflation Factor (VIF), we construct a classifier after reducing features. VIF quantifies the severity of multicollinearity, that is, a phenomenon in which one predictor variable in a multiple regression model can be linearly predicted from the others accurately.

Here, VIF is one of the indicators used to evaluate *multicollinearity*. In this paper, in order to make VIF of each feature 6 or less, experiments were conducted on a subset with reduced features. Based on the experimental results using all features, we calculate VIF and delete features with 6 or more VIF. To calculate VIF, statsmodel [36] was used.

TABLE II shows the dataset in this paper. We select events of 16 seismic sources whose observed number of moonquake events is 70 or more.

In this paper, the precision ratio, recall ratio, and F-value are used as indexes for evaluating the performance of classification of seismic sources.

The precision ratio is an index for measuring the accuracy of the classification, and the recall ratio is an index for measuring the coverage of the classification. F-value is the harmonic mean of recall and precision ratios and is an index in consideration of the balance of precision and recall. The score of the classifier in this paper is the average value of the F-values of the two classes targeted by the classifier.

D. Experiment Results

1) Experimental results using all features

a) Classification performance

Figure 3 is the average of the F-measures of classifiers for each seismic source. F-measure is the harmonic mean of precision and recall in statistical analysis. The vertical axis and the horizontal axis show seismic sources, each value is a score of the average of F-measure of classifier. In Figure 3, the highest classification performance is 0.96 and it is observed in multiple pairs of seismic sources. Also, the lowest classification performance is 0.54 as of classifier between A9 and A25. Figure 3 shows that some classification is difficult depending on combinations of seismic sources. Also, the number of classifiers with 0.9 or higher as classification performance is 20, about 17% of the total number of the classifiers. The number of classifiers with 0.8 or more and less than 0.9 is 60, 50% of the total. The number of classifiers with performance below 0.6 is only one. Most of the classifiers show high classification performance and show that the positional relationships of the planets are effective for the seismic source classification of the deep moonquakes.

b) Contribution ratio of features

Figure 4 shows the average value of contribution ratios for each feature. All features with the higher contribution ratios are those of the earth when they are calculated as the moon as the origin of the coordinate system. In addition, it shows that the contribution ratios of Jupiter 's features are high when the moon is the origin. By comparing features when the moon is the origin and when the earth is the origin, the features with the moon as the origin has a higher contribution ratio than the features with the earth as the origin. These observations suggest that the tidal forces are among the causes of moonquakes. Figure 4 indicates that relationships between the moon and the Earth affect the classification most strongly. However, there is a possibility that correlation between features, then it is necessary to further analyze each feature from view point of mutual independence. Therefore, in the following subsection, considering the correlations between features, we will describe the experimental results after feature reduction using VIF.

2) Experimental results of feature reduction using VIF.

a) Classification performance

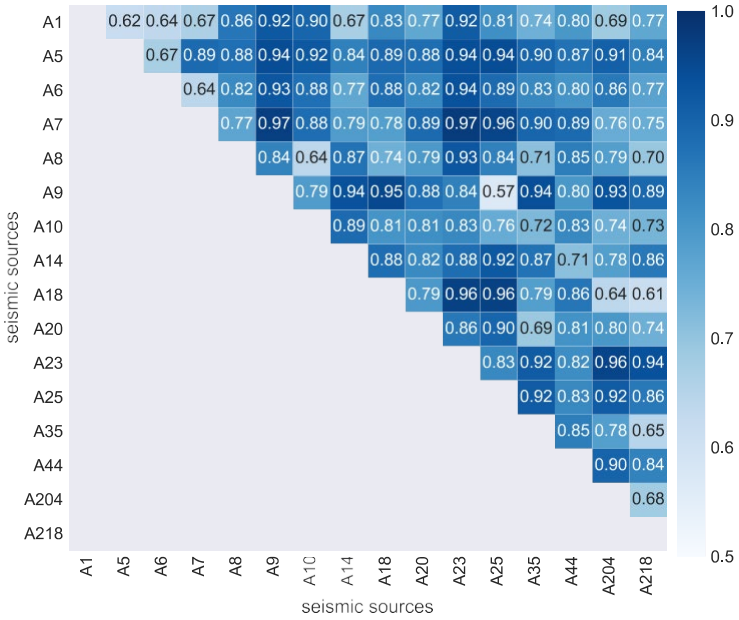
Figure 5 shows the average of the F-measures of the classifier when the features are reduced. Similarly, as in Figure 3, the vertical axis and the horizontal axis are seismic sources, respectively, and each value is the score of the F-measure of the classifier in Figure 5. In addition, the number of classifiers whose classification performance is 0.9 or higher is 26, about 22% of the total. 54 classifiers with 0.8 or higher but less than 0.9 are 45% of the total. There is one classifier whose classification performance is less than 0.6. Compared with Figure 3, these show that the classification performance does not change significantly.

b) Contribution ratio of features

Figure 6 shows the average value of the contribution ratios of each seismic source after feature reduction. After reducing features, earth features when the origin is the moon are reduced to 4 features of the top 10 features which existed before feature reduction. The four features between top 11 and 14 positions of the features of Jupiter when the origin is the moon, as shown in Figure 4, are reduced to one feature. Other parameters of Jupiter are thought to have been affected by other features. The subset of the features after feature reduction is considered to have small influence of multicollinearity. Therefore, there is a possibility that the features of the Earth and some of the features of Jupiter are effective for classification when the moon is the origin. These results are microscopic explanations made directly from the model constructed by Balanced Random Forest.

E. Discussion of methods and features

By using Balanced Random Forest, contribution ratios of features can be easily calculated in addition to classification performance, so it is useful for feature analysis like the scientific research described in this section. However, in this method, there is room for consideration of parameters of classification techniques depending on the seismic sources as the classification targets. Moreover, in order to obtain higher classification performance, it is necessary to consider many classification methods. Furthermore, it is necessary to apply a method considering waveform information simultaneously collected by the NASA Apollo project. In addition, since the



findings obtained in this paper are only correlations, it is difficult to directly estimate the causal mechanism of the deep moonquakes. However, the results of this paper are shown to be useful for further analysis and knowledge creation by experts. If the knowledge of experts such as the physical mechanism about moonquakes is available, the elucidation of the causal relationships between the seismic sources and the planetary bodies and ultimately that of the causal mechanism of the moonquakes (possibly related to tidal forces) can be expected. In general, expertise in any domain is expected to increase our understanding of the causal relationships suggested by our correlation analysis.

Figure 5. Averages of F-measures for pairs of seismic sources after feature reduction.

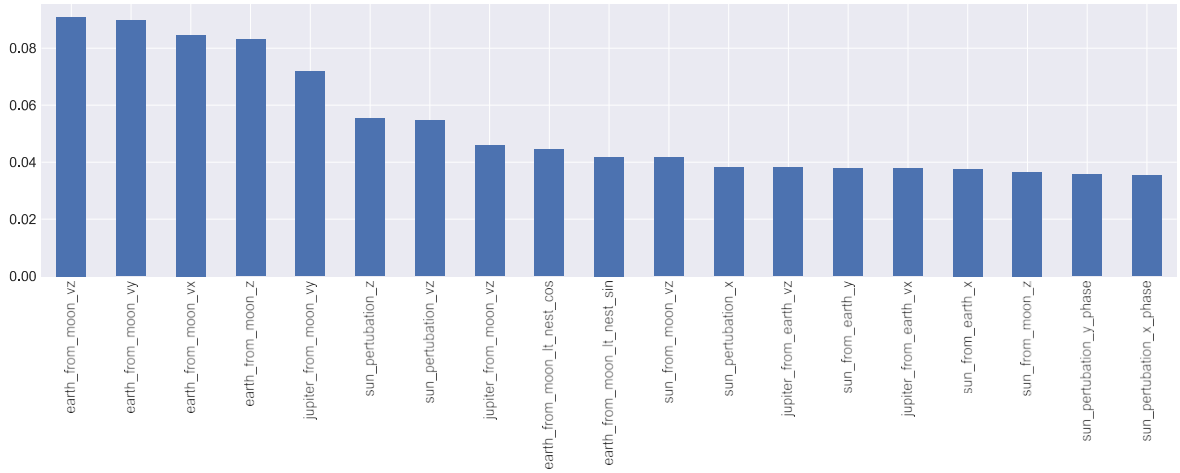


Figure 6. Averages of contribution ratios for each feature after feature reduction.

V. CASE STUDY: MICRO EXPLANATION FOR CENTRAL PEAK CRATER APPLICATION

First, we will summarize this case as follows.

a) *(Scientific and explanatory objectives) In this case, also related to lunar and planetary science, in order to understand the internal structure and movement of the moon, it is conceivable to use the materials inside the moon as a clue. The central hill in the crater is attracting attention as a place where the materials inside the moon are exposed on the moon surface. However, not all craters with central hills on the moon have been identified. Therefore, it is scientifically necessary to make the catalog. Therefore, it is also necessary to explain to the relevant scientists the grounds for judging the craters included in the found candidates as craters with central hills.*

b) *(Data used for use case) We used about 7,200 images provided by both NASA and JAXA. Each image has been resized to 512 (height) * 512 (width) * 1*

(normalized elevation). We divided the whole images into equal numbers of images with three labels, that is, craters with central hills (central-hill craters), craters without central hills (normal craters), and non-craters.

c) *(Methods used for use case) We used RSPD [37] to detect craters only for preparation of training data of CNN [14][38]. Next, we applied learnt a CNN to find central-hill craters including unknown and known ones and used Grad-CAM [15] to know the evidence for judging central-hill craters.*

d) *(Result) We could classify three classes with 96.9 % accuracy, which was verified by the scientific members of our research team. We also could show the scientific members of our research team individual evidences for judging central-hill craters consisting of crater rims and central hills.*

Next, we will describe this case in more depth for the explanation functions although we have already introduced it to explain the way of model construction in our previous work [3].

A. Discovery of central peak craters

Scientific data are a kind of real-world data. By taking an example of research conducted by our team including JAXA researchers using scientific data which are also open data, we explain integrated analysis [39]. We use hypothesis generation based on differences between original data and their rotations.

A detailed map of the surface of the moon was provided by JAXA launched lunar orbiter KAGUYA (SELENE) [40]. Of course, KAGUYA's purpose goes beyond making a map of the moon. The goal is to collect data that will help elucidate the origin and evolution of the moon. In order to further pursue such purposes, it is important to examine the internal structure of the moon.

One of the methods to examine the internal structure of the moon is to analyze the data of moonquakes (i.e., corresponding to earthquakes) that occur in the moon. Research is also being conducted to classify the hypocenters of moonquakes based on the data of moonquakes provided by the NASA Apollo program. Among these studies are our research which showed that it is possible to classify moonquakes by features such as the distance between the moon and the planet alone without using seismic waves themselves as described in Section IV.

Another method is to launch a spacecraft to directly explore the internal structure of the moon. However, it is not sufficient to land the spacecraft anywhere on the moon. That is naturally because there are limited resources such as budgets that can be used for lunar exploration. In other words, it is necessary to determine the effective point as the target of the spacecraft based on the evidence. This way is an example of EBPM, making an effective plan based on evidence under limited resources.

On the other hand, whether large or small, a lot of craters exist in the moon. Among them, special crater with a structure called "central peak" (hereinafter referred to as "central peak craters") is present (see Figure 7). The central peak is exposed on the moon and lunar crustal substances are also exposed therein. Therefore, it is likely that central peak craters scientifically have important features. In other words, the exploration of the surface of the central peak makes it possible to analyze the surrounding internal crustal materials in a relatively easy way. By this, it is expected that not only the origin of the crater and central peak can be estimated, but also the surface environment of the past lunar surface and the process of crustal deformation of the moon can be estimated.

But with respect to the central peak crater as the exploration target, conventionally the confirmation of existence of the central peaks has been visually done by the experts. So, the number of craters known as centralpeak craters is rather small. This problem can be solved by automatic discovery and cataloguing of central peak craters to significantly increase the number of central peak craters as candidate exploration points.

Thus, in this case with creating the catalog of central peak craters as our final goal, a specific technique for automatic discovery of central peak craters has been proposed. This case uses DEM (Digital Elevation Model) of the lunar surface as results observed by the lunar orbit satellite "KAGUYA" of JAXA [40]. Paying attention to the image characteristics of DEM, we apply CNN (Convolutional Neural Network [14] [38]) as a popular technique for deep learning, which is recently in the limelight as AI, to construct the discrimination model. We evaluate discriminability of the central peak crater by the model by experiments.

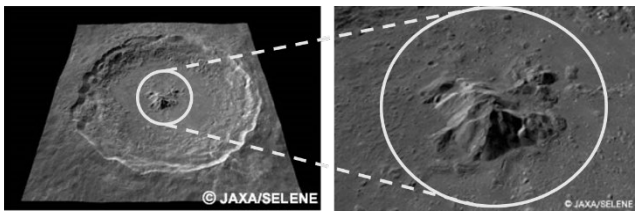


Figure 7. Example of central peak crater.

B. Integration Hypothesis

The central peak crater is identified by the following two-step procedure.

1) Crater extraction on the moon by RPSD method

Craters are extracted by using the popular and secure method called RPSD (Rotational Pixel Swapping for DTM) for digital terrain models. Here, DTM (Digital Terrain Model) is a digital terrain model similar to the digital elevation model(DEM). The RPSD method focusses on the rotational symmetry when rotating the image of the DEM at a certain point (i.e., central point). That is, RSPD uses the fact that the negative gradient property from the rim of the crater to the center in the lunar DEM does not change with rotation of craters. In other words, we make the difference between the original candidate crater and the rotated one (corresponding to the difference in the observation mode for the same object) and confirm that the feature about rotational symmetry does not change in order to discriminate craters. In a word, this method corresponds to our generalized difference method in which hypotheses (craters) are found by focusing on differences obtained by different means for the same object (candidate craters).

2) CNN-based automatic discrimination of central peak crater from extracted craters

In general, in the discrimination phase for each layer of deep learning, each output node multiplies the input values by weights, takes their sum, and adds their bias to the sum, and then outputs the result in the forward direction.

In the learning phase of deep learning, as a problem of minimizing the error between the output of discrimination and the correct answer, the values of weights and biases are updated by differentiating the error function with respect to the weight and bias of each layer.

C. Integrated Analysis

First, using RPSD, we extract the DEM data of each candidate crater and provide them with a label (non-crater, non-central peak crater, and central peak crater) to create training data. We learn CNN model thus using the training data and discriminate the central peak craters by using the CNN model. From recall ratios obtained by experiments focusing on how much correct answers are contained in the results, the possibility that CNN is an effective technique in the central peak crater determination is confirmed.

In order to confirm reasons for the classification results, we visualize the contribution areas in input images which affect the model (i.e., individual evidence).

We use Grad-CAM [15], a method for visualizing contribution areas for each label in an input image. We use it because it has an affinity for CNN.

The left part (see Figure 8) is an input image, the central part is the contribution area for "central-peak crater" label, and the right part is the contribution area for "normal crater" label.

The contribution area for "central-peak crater" has heating area inside the crater, and heating area is covering a central peak. On the other hand, the central peak area does not have heating area at the contribution area for "normal crater." Therefore, we think that the central peak area contributes classification for "central-peak crater" label. Thus, the contribution areas as the micro explanation can help the scientists to understand the corresponding classification results.

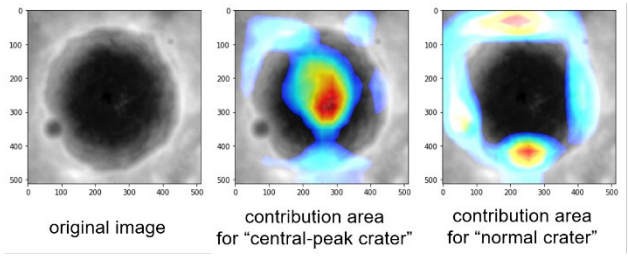


Figure 8. Explanation of individual evidences.

VI. CONCLUSION

In this paper, we proposed a general framework of explanation necessary to widely promote implementation of analytical applications using SBD. The procedure of a tourism application based on integrated data model was described as an example of a macro explanatory function. In addition, we used Balanced Random Forest as a micro explanatory function to extract features effective for the seismic source classification of the deep moonquakes from the temporal and spatial features of the planets. We described another example of a microscopic-explanatory function to explain individual evidences for discrimination of central peak craters.

The results of social science research (i.e., an example of macro explanation of procedures) were explained to external travel experts to confirm their effectiveness. Regarding to the explanation of scientific results (i.e., examples of micro explanations of model features and judgements), we have positive feedbacks from the relevant scientists in our research team based on the scientific effectiveness.

We reiterate the whole process of the SBD application with explanations as the summarization of the contribution of this paper.

1. *The user describes the procedure for data management and model generation by utilizing the data model (i.e., SBD model) and the hypothesis generation methods (e.g., generalized difference method).*
2. *The macroscopic-explanation function uses the constructed description for the explanation.*
3. *The microscopic-explanation function finds the effective model features and individual judgement basis by executing the constructed model using the explanation-oriented techniques (e.g., Balanced Random Forest and Grad-CAM).*

We will continue to verify the versatility of the explanatory framework by applying it to a wider variety of use cases in the future. We will also continue to interview the parties concerned and listen to the opinions of experts at international conferences on the effectiveness of the framework for explanation. In fact, we have already presented the candidate list of central hill craters with the micro explanations to the scientists in the related field. They have definitely found unidentified central hill craters among the candidates. As a result, a new project has recently been initiated to re-estimate the quantitative relationships holding between the radius of the central hill crater and the height of the central hill based on our findings.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number 16K00157, 16K16158, 20K12081 and Tokyo Metropolitan University Grant-in-Aid for Research on Priority Areas Research on social big data.

REFERENCES

[1] H. Ishikawa, Y. Yamamoto, M. Hirota, and M. Endo, “Towards Construction of an Explanation Framework for Whole Processes of Data Analysis Applications: Concepts and Use Cases,” Proc. The Eleventh International Conference on Advances in Multimedia MMEDIA 2019 (Special tracks:SBDMM: Social Big Data in Multimedia), Mar. 2019.

[2] H. Ishikawa, Social Big Data Mining, CRC Press, 2015.

[3] H. Ishikawa, D. Kato, M. Endo, and M. Hirota, “Applications of Generalized Difference Method for Hypothesis Generation to Social Big Data in Concept and Real Spaces,” Proc. the 11th International Conference on Management of Digital EcoSystems (MEDES 2019), pp. 44–55, November 2019. https://doi.org/10.1145/3297662.3365822 [retrieved: May, 2020].

[4] EU GDPR, https://eugdpr.org/ [retrieved: May, 2020].

[5] J. Hemmi, “Japan's 'information banks' to let users cash in on personal data,” NIKKEI Asian Review, May 09, 2019.

[6] K. Kato, R. Yamada, Y. Yamamoto, M. Hirota, S. Yokoyama, and H. Ishikawa, “Analysis of Spatial and Temporal Features to Classify the Deep Moonquake Sources Using Balanced Random Forest,” Proc. The Ninth International Conferences on Advances in Multimedia (MMEDIA 2017), April 2017.

[7] T. Tsuchida, D. Kato, M. Endo, M. Hirota, T. Araki, and H. Ishikawa, “Analyzing Relationship of Words Using Biased LexRank from Geotagged Tweets,” Proc. ACM MEDES International Conference, pp. 42-49, 2017.

[8] C. Chen, A. Liaw, and L. Breiman, “Using random forest to learn imbalanced data,” University of California, Berkeley, pp. 1-12, 2004.

[9] H. Ishikawa, “Object-oriented database systems,” (C. T. Leondes ed.) Database and Data Communication Network Systems Techniques and applications, vol. 1, pp. 77-122, Academic Press 2002.

[10] D. Smith, R. St. Andre, and M. Eggen, A Transition to Advanced Mathematics, Brooks/Cole Pub Co., 2014.

[11] R. D. Peng, “Reproducible Research in Computational Science,” Science, vol. 334, issue 6060, pp. 1226-1227, 2011.

[12] K. McCain, “Explanation and the Nature of Scientific Knowledge,” Sci & Educ, vol. 24, pp. 827–854 (2015). https://doi.org/10.1007/s11191-015-9775-5 [retrieved: May, 2020].

[13] H. Ishikawa, D. Kato, M. Endo, and M. Hirota, “Generalized Difference Method for Generating Integrated Hypotheses in Social Big Data,” Proc. ACM MEDES International Conference, pp. 13-22, 2018.

[14] X.-W. Chen and X. Lin, “Big Data Deep Learning: Challenges and Perspectives,” IEEE Access, vol. 2, pp. 514-525, 2014.

[15] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization,” Proc. The IEEE International Conference on Computer Vision (ICCV), 2017, pp. 618-626 https://arxiv.org/abs/1610.02391 [retrieved: May, 2020].

[16] W. V. Siricharoen, “Infographics: The New Communication Tools in Digital Age,” Proc. International Conference on E-Technologies and Business on the Web, pp. 169-174, 2013.

[17] Z. C. Lipton, “The Mythos of Model Interpretability,” Communications of the ACM, vol. 61, no. 10, pp. 36-43, October 2018.

[18] F. D. Velez and B. Kim, “A roadmap for a rigorous science of interpretability,” pp. 1-13, 2017 (arXiv: 1702.08608, 2017).

[19] D. Martens, B. Baesens, T. V. Gestel, and J. Vanthienen, “Comprehensible credit scoring models using rule extraction from support vector machines,” Rule extraction from support vector machines, pp. 33-63, 2008.

[20] L. A. Hendricks, Z. Akata, M. Rohrbach, J. Donahue, B.

Schiele, and T. Darrell, "Generating visual explanations," Proc. European Conference on Computer Vision, pp. 3-19, Springer, 2016.

[21] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. Mueller "How to explain individual classification decisions," The Journal of Machine Learning Research, vol. 11, pp. 1803-1831, August 2010.

[22] Z. Si and S. C. Zhu., "Learning and-or templates for object recognition and detection," IEEE Trans. Pattern Anal. Mach. Intell., vol. 35, no. 9, pp. 2189-2205, 2013.

[23] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," Science, vol. 350, issue 6266, pp. 1332-1338, 2015.

[24] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," Proc. CHI 2016 Workshop on Human Centered Machine Learning, pp. 1135-1144, 2016 (arXiv: 1602.04938v1 [cs.LG] 16 Feb 2016).

[25] M. Du, N. Liu, and X. Hu, "Techniques for Interpretable Machine Learning, " Communications of the ACM, vol. 63, no. 1, pp. 68-77, 2020.

[26] K. W. Su, C. L. Liu, and Y. W. Wang, "A principle of designing infographic for visualization representation of tourism social big data." Journal Ambient Intell. Human Comput, pp. 1-21, 2018, doi:10.1007/s12652-018-1104-9.

[27] OECD, "EBPM, Evidence in Education: Linking Research and Policy." http://www.oecd.org/education/ceri/38797034.pdf [retrieved: May, 2020].

[28] Flickr https://www.flickr.com/ [retrieved: May, 2020].

[29] Twitter https://twitter.com/ [retrieved: May, 2020].

[30] M. Hirota, K. Saeki, Y. Ehara, and H. Ishikawa, "Live or Stay?: Classifying Twitter Users into Residents and Visitors," Proc. International Conference on Knowledge Engineering and Semantic Web (KESW 2016), pp. 1-2, 2016.

[31] K. Mitomi, M. Endo, M. Hirota, S. Yokoyama, Y. Shoji, and H. Ishikawa, "How to Find Accessible Free Wi-Fi at Tourist Spots in Japan," Volume 10046 of Lecture Notes in Computer Science, pp. 389-403, 2016.

[32] DARTS Available: http://darts.jaxa.jp [retrieved: May, 2020]

[33] L. Breiman, "Random forests," Machine learning, vol. 45, no. 1, pp. 5-32, 2001.

[34] NAIF, https://naif.jpl.nasa.gov/naif/ [retrieved: March, 2019].

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnayet, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[36] S. Seabold and J. Perktold. "Statsmodels: Econometric and statistical modeling with python," Proc. 9th Python in Science Conference, pp. 57-61, 2010.

[37] S. Yamamoto, T. Matsunaga, R. Nakamura, Y. Sekine, N. Hirata, and Y. Yamaguchi, "An Automated Method for Crater Counting from Digital Terrain Model Using Rotational Pixel Swapping Method," Proc. 49th Lunar and Planetary Science Conference, 2 pages, 2018.

[38] Y. LeCun, Y. Bengio, and Geoffrey Hinton, "Deep learning," vol. 521, pp. 436-444, 2015. doi:10.1038/nature14539

[39] S. Hara H. Inoue, M. Yamamoto, Y. Yamamoto, M. Otake, H. Otake, T. Araki, M. Hirota, and H. Ishikawa, "Automatic Extraction of Lunar Central Peak Craters by Deep Learning," 16th AOGS Annual Meeting, 2019.

[40] JAXA, KAGUY(SELENE) Data Archive https://darts.isas.jaxa.jp/planet/pdap/selene/ [retrieved: May, 2020].

An Algorithmic Solution for Adaptable Real-Time Applications

Lial Khaluf
Email: lial.khaluf@googlemail.com

and

Franz-Josef Rammig
Email: franz@upb.de
University of Paderborn
Paderborn, Germany

Abstract — **Most real-life facilities nowadays belong to real-time systems. E.g., airplanes, trains, cars, medical machines, alarming systems and robotics, etc. Some of these systems behave on their own, separately or in cooperation. Some of them interact with humans. Operating and interaction is done under conditions defined inside the systems and the environment. However, these systems and environments might grow or change over time. In order to develop safe and high-quality real-time applications, we need to make them highly self-adaptable systems. In this paper, we describe the detailed steps of a real-time aware adaptation algorithm and we go through the boundedness proof of each step. The algorithm mimics the behaviour of organic cells. It introduces a new kind of real-time tasks. This kind enables tasks to change their behaviour at run time according to internal changes inside the system and external changes in the environment, preserving all real-time constraints. This includes real-time constraints for the adaptation process itself. Following this concept, real-time tasks are turned to be real-time cells.**

keywords – adaptation array; genetic optimization; organic behavior; boundedness.

I. INTRODUCTION

Current trends of adaptation mechanisms have been applied in traditional software systems as well as real-time (RT) systems. RT systems, however, lack the required flexibility for adapting themselves to changes being unpredictable at design time. In this paper, we try to overcome this limitation by providing an adaptation solution at run time. The problem we solve assumes a system that has to fulfil a set of hard-deadline periodic and aperiodic tasks. Aperiodic tasks might have dependencies between each other. Modifications to the current set of tasks may happen at run time as, e.g., a result of environmental changes. Modifications may include adding a new task or a set of dependent tasks, updating a task or a set of dependent tasks, and deleting a task or a set of dependent tasks. We assume that by updating dependent tasks, no new tasks are added to the dependability graph, and no existing tasks are deleted. The goal of the approach presented in this paper is to adapt the newly arrived modifications at run time without breaking any of the stated RT constraints.

In [1], we have introduced a summary of the adaptation algorithm. In this paper, we go through details of it, and prove the boundedness of each step. In our solution, we assume a bounded, but online expandable, ecosystem of RT tasks. Each RT task may exist by means of a bounded, but online expandable, set of variants. All variants of a task share the same principal functionality, however, at different quality levels. Whenever activated, the algorithm tries to find a solution that can accept all currently requested modifications. For this reason, it tries to find a selection of task variants such that all RT constraints are satisfied and at the same time the overall quality over all tasks is maximized. The underlying ecosystem is represented by a two-dimensional data structure called $RTCArray$ [2]. It is divided into classes. Each class is represented by a column. A class represents a unique functionality. Each variant within a column provides the same principal functionality as the other variants, but with different time and quality characteristics. We model quality by a cost function where cost is defined as the inverse of quality, i.e., highest possible quality is modelled by cost 0. We call a variant an RT cell. The reason is that the structure and behaviour of variants can change at run time following the environment of the system. Modification requests may arrive at any time. According to their priority, they are put into a queue with a predefined capacity. The queue is handled by a central cell called the Engine-Cell (EC). EC tries to find a best combination of variants such that the arrived modifications can be accepted. The problem of selecting a best combination of variants is mapped on solving a Knapsack problem, executed on a so-called $AdaptationRTCArray$. This is a subset of $RTCArray$ restricted on currently running cells that have to continue execution and augmented by newly arriving modifications. As this Knapsack problem has to be solved under RT constraints, an "anytime algorithm" is needed to solve it. We decided for a genetic algorithm with a trivial initial population. The individuals of populations are evaluated concerning their respected overall cost under the constraint that all RT constraints have to be satisfied. It is assumed that the RT system is running under Earliest Deadline First (EDF) algorithm [16] as principal scheduling algorithm. EDF is used to schedule a set of independent tasks by always giving the priority to the task with earliest absolute deadline [16]. As the challenge we are aiming to overcome in our approach considers also the case of dependent tasks, Earliest Deadline First with Precedence Constraints algorithm (EDF*) [5] is used. EDF* transforms a set of dependent tasks into a set of independent tasks by recalculating the timing parameters of the

tasks. The resulting independent tasks can then be scheduled by EDF. The dependent set is schedulable if and only if the independent set can be schedulable [16]. For considering aperiodic tasks, we use the Total Bandwidth Server (TBS) [6]. TBS calculates new absolute deadlines of aperiodic tasks, so that the server can behave as a periodic task, which is schedulable with the other periodic tasks in the system under EDF. In our approach, the RT constraints of all periodic tasks can be calculated using a utilization bound 1 and those for aperiodic tasks by checking that deadlines calculated by TBS are less or equal than specified hard deadlines.

In Section II, we present the related work. Section III presents the definitions of some basic concepts. Section IV includes a scenario example of an RT application, in which our approach can be applied. Section V describes the algorithm and boundedness proof of each step. In the last section, we present a conclusion and future work.

II. Related Work

In this paper, we are solving an optimization problem. The goal is to provide enough processor capacity for the current requests and the currently running cells while minimizing the costs. The problem can be modelled by a multidimensional multiple-choice knapsack problem (MMKP). Many approaches were defined for solving this problem. In [7] a metaheuristic approach is developed. It simplifies the MMKP into multiple-choice knapsack problem (MCKP) by applying a surrogate condition for cost. In the MCKP, there are several groups of items. It is required that one item is selected from each group, so that the total benefit is maximized without exceeding the capacity of the knapsack. For finding a feasible solution and enhancing it in a short time, the algorithm in [7] is considered to be a good choice. In our approach, however, we use a genetic algorithm. It can decide already in the first step whether a feasible solution exists or not. Enhancing the solution is bounded by a specific time.

In [8] a heuristic algorithm is used for solving the MMKP by using convex hulls. The idea is to simplify the MMKP into MCKP. This is done by multiplication of a transformation vector. Once the MCKP is constructed, each group of items can be represented on X-Y Axes. X represents the resources used by the items. Y represents the benefit that should be maximized. An initial solution is found by selecting an item from each group. The selected item is the one with lowest benefit. After that, three iterations are done. In each iteration, the penalty vector is used to turn each of the resource consumption vectors into a single dimension vector. The frontier segments of the items are calculated. “A segment is a vector with two items representing a straight line.” [8]. According to the angle of the segment, it is ordered within the list of segments. The segments should be put in a descending order. For each segment, P1 and P2, the items associated with the segment, are considered. A current solution is calculated by selecting the item associated with P1 and the same is applied for P2. If utility of the current solution is smaller than the utility of the saved solution, the saved solution is kept. Penalty is adjusted for the next iteration. After iterations are done, if the current solution is not feasible, then no solution is found, else the current solution is set to be the final solution. Following this method requires the values and weights to be known before penalty vectors and convex hulls are constructed. In our approach this is not possible because a pre-knowledge of members to be selected in each group are required to calculate values of weights. The reason is that one of the considered weights requires for its calculation the deadline, which can be calculated according to TBS [6]. The calculation of a deadline, which belongs to a selected item in a group depends on the deadlines of selected members in previous groups.

In [10], a reduce and solve algorithm is used for solving MMKP. The approach depends on group fixing and variable fixing to reduce the problem. Then it runs CPLEX [14] to solve the reduced problem. Also here it is required to know the benefits and weights before start solving. In our approach this is not possible.

In [9], three algorithms are introduced to solve MMKP. The first algorithm tries to find an initial solution for the guided local search. It is applied by assigning a ratio for each item in the groups. The ratio is the value divided by the Scala Product of the weight of the item, and total capacity of the knapsack. Items, which own best ratios are selected. If a feasible solution is not reached, then an item with heaviest ratio is chosen to be swapped with another item from the same group. If this does not result in a feasible solution, then the lightest item from the same group is selected. This iteration continues until a feasible solution is found. The initial solution can be enhanced by applying the second algorithm, a complementary constructive procedure (CCP). It consists of two stages. The first stage swaps selected items with other items within their groups to enhance the already found feasible solution. If the resulting solution is feasible then the swapping is considered to be valid. The second stage replaces the old item by the newly selected one. The third algorithm is the derived algorithm. It starts by applying the constructive procedure of the first algorithm in order to get an initial feasible solution. If the solution cannot be improved by CCP, a penalty parameter is applied to transform the objective function, and a new solution is acquired by CCP. If the new solution achieves improvement, then a normalization phase is applied to get the original values of profits. If it does not achieve any improvement, then a normalization phase is applied and a penalty is applied again. The iteration continues until a stopping condition is reached. In the previously three described algorithms, it is required to know the benefits and weights before start solving. In our approach this is not the case.

In [11], a reactive local search algorithm to solve MMKP is presented. A constructive procedure (CP) and a complementary procedure CCP are used to acquire an initial solution. CP uses a greedy procedure to acquire the initial solution. CCP enhances the solution acquired by CP. The enhancement is done by following an iterative approach that swaps elements in the same class (group). The reactive local search (RLS) is applied to enhance the solution acquired by CCP. RLS consists of two procedures. The first one is called degrading strategy, and the second one is called deblock procedure. The degrading strategy consists of three steps. It selects an arbitrary class, changes arbitrary elements inside it if this exchange will result a feasible solution, repeats steps 1 and 2 several times, and terminates with a new solution. The deblock strategy starts by constructing a set of elements. Each element consists of two classes. The strategy runs a loop on the set until no element exists in it. In each run of the loop, it investigates if there is a couple of items in both chosen classes, so that the objective

value of resulting feasible solution is better than the previous solution value. The resulting feasible solution considers the couple of items in chosen classes, in addition to the fixed items, which belong to the classes other than the chosen ones. The deblock strategy exits with the best solution. [11] describes also the modified reactive local search algorithm (MRLS). It replaces the deblock procedure by a memory list to enhance computation time. In RLS and MRLS, again it is required to know the benefits and weights before start solving. In our approach this is not the case.

Evolutionary algorithms have been studied in a couple of previous works for solving different kinds of knapsack problems. For example, in [12], a general approach of using genetic algorithm to solve MMKP is described. It starts by selecting an initial population. This can be done randomly. The fitness of the population is evaluated according to the fitness function. The fitness function is defined according to the objective function of the knapsack problem. Then a loop runs until a predefined condition is satisfied. In each iteration, a new population is selected from the previous one using the roulette wheel selection [15]. Crossover and mutation are applied. The resulting population is evaluated. The current generation is combined with previous one. Finally, the resulting individuals are ordered to find a best solution. The algorithmic principle in [12] is similar to our approach, however, the selection process differs from the selection process in our approach. In our approach, we can determine if a feasible solution exists once we construct the first individual. Further steps work only on optimizing the solution. In [12], it is not explained if one can recognize the existence of a feasible solution once the first individual is constructed.

In [13], an evolutionary algorithm is used to solve MMKP. The idea is to solve a manufacturing problem. Operators should be distributed among machines to process components of products. This has to be done in an efficient way to keep work hours within a predefined limit. Operators differ regarding their experience or working ability. To model this situation, binary coded chromosomes are constructed. Each chromosome has three dimensions: machines, operators and components. Chromosomes are transferred into two dimensional chromosomes. A first generation of chromosomes is chosen. It should contain only feasible solutions. A loop is run on generations. In each run, a new generation is generated by applying selection and mutation on the previous generation. The loop continues until a predefined condition is satisfied. The algorithmic principle in [13] is similar to our approach, however the principle of setting the fitness function and the principle of the selection process differ from our approach. In our approach, we can determine if a feasible solution exist once we construct the first individual. Further steps work only on optimizing the solution. In [13], it is not explained if one can recognize the existence of a feasible solution once the first individual is constructed.

The comparison in this section between our approach and the previously introduced approaches points out that our approach provides more flexibility in terms of solving a broader set of problems where parameters can be calculated at runtime, and the existence of a feasible solution can be known in a very early stage of the algorithm.

In the next section, we present the basic concepts of RT operating systems, the knapsack problem, and the genetic algorithms.

III. Basic Concepts

RT systems are computing systems, in which the correctness of behavior depends not only on the computation results but also on the response time. “Examples of RT systems could be automotive applications, flight control systems, robotics, etc” [16].

- A real-time task: is characterized by many properties. In the following, we mention some of them:

1) Arrival time (a): it is the time at which the task is released and becomes ready for execution, called also release time. [16]
2) Execution time (C): is the time required to execute the task without interruption. [16]
3) Absolute deadline (d): is the time that the task execution should not exceed. [16]
4) Relative deadline (D): is the time difference between the absolute deadline and the arrival time. [16]
5) Start time (s): is the time of starting the execution of a task. [16]
6) Finishing time (f): is the time of finishing the execution of a task. [16]
7) Criticalness: is a parameter that indicates whether the task is hard or soft. [16]

- A soft RT task: is a task that does not cause a catastrophic result when its deadline is not met, but might cause a decrease in the performance. [16]

- A hard RT task: is a task that may cause catastrophic results in the system environment if its deadline is not met. [16]

- A periodic task: is a task that is activated in regular time periods, where each activation is called an instance of the task. [16]

- An aperiodic task: is a task that is activated in irregular time periods, where each activation is called an instance of the task. [16]

- Precedence constraints: represent the precedence order that tasks might have to respect concerning the order of their execution. Precedence constraints are normally represented by a directed acyclic graph (DAG) [16]. DAG has no directed circles [17].

The knapsack problem is an NP-hard problem. In this problem, there is a set of items, where each item has a benefit and a weight. A subset of items should be selected, so that the sum of their benefits is maximized, and the capacity of the knapsack is not exceeded [18]. There are several types of knapsack problems [18]:

1) 0-1 Knapsack problem: There is a set of items. We want to put the items into a knapsack of capacity W. We should pick a set of mutually different items, so that the total value is maximized and the capacity of the knapsack is not exceeded.
2) Bounded Knapsack problem: Same as 0-1 knapsack problem. However, we can select more than one instance from each item. The number of selected instances is limited by a certain bound specified for each item.
3) Multiple knapsack problem: Same as 0-1 knapsack problem. However, here we have more than one knapsack. Each knapsack has a capacity. The knapsacks

should be filled with the items, so that the total value is maximized, and the capacity of each knapsack is not exceeded.

4) Multiple-choice knapsack problem: Same as 0-1 knapsack problem. However, the items should be chosen from different disjoint classes. Only one item is chosen from each class.
5) Multidimensional multiple-choice knapsack problem: Same as multiple-choice knapsack problem. However, the knapsack may have a vector of capacities. Each capacity represents the availability of different resources (dimensions) that the knapsack provides. The weight of each item is represented by a vector. Each weight in the vector reflects the weight of a unique resource. When a set of items is chosen by solving the knapsack problem, the sum of weights for a specific resource should not exceed the resource capacity provided by the knapsack.

Evolutionary algorithms are heuristics that aim to find a best solution. An evolutionary algorithm starts with an initial population. The population consists of several individuals. Each individual is characterized by a fitness value. The individuals with best values are selected to reproduce new individuals, as illustrated by Figure 1. [19]

A genetic algorithm is a type of evolutionary algorithms. It consists of the following steps: Initial population, evaluation, fitness assignment, selection, and reproduction. [25]

1) Initial population: In this step the initial population is chosen. The initial population consists of individuals. [25]
2) Evaluation: Evaluates the current population according to an objective function. [25]
3) Fitness assignment: Determines the fitness of the population. [25]
4) Selection: Selects the fittest individuals for the reproduction process. [25]
5) Reproduction: Applies crossover and mutation to generate new individuals. [25]

The principle is to reach an optimal solution. Reaching this solution is done by searching the design space to find an initial population. The individuals of this population are tested according to an objective function. New generations are then produced from the current generation by applying selection, crossover and mutation. An individual may be a number, set of integers, two dimensional or three dimensional variable, etc. Finding the initial population could be done randomly, by going through an algorithm, or other methods.

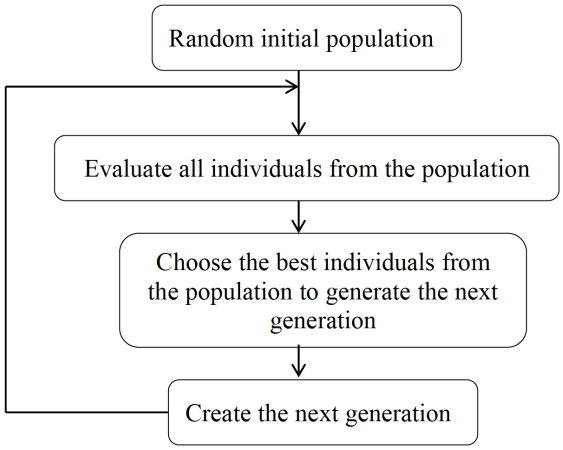


Figure 1. Evolutionary Algorithms. [19]

Boundedness is equivalent to the fact that the algorithm terminates after a finite time whenever being started. To guarantee boundedness, the following conditions should be satisfied:

1) There must be no external influences which are not under control of the algorithm.
2) There must be no deadlocks and no unbounded blocking.
3) There must be no while/until loops which are not terminating: A classical test in this case is checking whether the function to be calculated is bounded, monotonic and not asymptotic. If these three conditions are true then we are sure that the respective loop will terminate.

In the following, we present the definition of bounded, monotonic and asymptotic functions:

Bounded functions: "A function is bounded from below if there is k such that for all x, $f(x) \geq k$. A function is bounded from above, if there is K such that, for all x, $f(x) \leq K$." [20]

Monotonic functions: " A function is monotonically increasing if for all x and y, such that $x \leq y$ one has $f(x) \leq f(y)$, so f preserves the order. Likewise a function is called monotonically decreasing if whenever $x \leq y$ then $f(x) \geq f(y)$, so it reverses the order " [21]

Asymptotic functions: "A function that increases or decreases until it approaches a fixed value, at which point it levels off," (when x tends versus infinity). [22]

In the next section, we introduce the robotic surgical system as an appropriate example for an RT application, where our approach can be applied.

IV. Scenario

Let us assume a telerobotic surgery system [23], where the surgeon is performing the surgical operations remotely with the help of a robotic surgery system, a set of surgical instruments, a set of endoscopic tools, a set of medical, technical, and energy resources, and a deterministic network. The surgical operations are taking place online, where the surgeon deals with the digital extension of the patient and the patient is operated by the digital extension of the surgeon. The surgeon side is the Master side. The patient side is the Slave side. See Figure 2. On the Master side, the robotic surgical system provides a vision system that translates the information coming from the Slave side. On the Slave side, the system provides a controller which translates the decisions coming for the Master side into instructions to be applied by the robotic arms, endoscopic tools and other instruments which will in turn act as a digital extension of the surgeon. The ability of the system to adapt itself to the evolutions of surgical actions is limited by the surgeon 's ability to react to these evolutions with the required speed so that the operation is performed successfully.

To overcome this limitation, we assume that the surgeon is only responsible for deciding which surgical actions should

take place during the operation. However, the actions steps and characteristics are predefined and performed by the system and according to the system parameters. The previous assumption defines the surgical operation to be a set of surgical actions that are triggered online and must be accomplished in real-time. This set should be able to change its structure and behavior at run-time to enable the system to adapt itself to environmental changes on the Slave side. The adaptation process should preserve all RT constraints. Here, the overhead imposed by the adaptation process itself has to be considered as well. To perform the surgical operation successfully, the robotic surgical system collects all internal and external parameters that reflect the environment state on the Slave side. The parameters are then analyzed by the system on the Master side and represented using a vision system. This enables the surgeon to read the current state of the patient and to decide if a new surgical action should take place or a currently running surgical action should be updated. The decision is then studied by the system to see whether it has influence on meeting the real-time constraints of the surgical operation. If no negative influence exists, the decision is applied to the Slave side. However, if this is not the case, an adaptation algorithm is run to check whether there is a possibility to change the structure and behavior of the current surgical actions set in a way that enables to apply the decision and preserves all real-time constraints. If this succeeds, the set is modified and the decision is applied. Otherwise, the surgeon is informed about the necessity to make another decision. The measurements of the patient as e.g. pressure, temperature, view of the surgical field, etc. represent the internal parameters. The measurements of the environmental factors as e.g. the energy sources including light, temperature, etc. of the surgical room, and the measurements of other resources as e.g. number and kinds of surgical instruments, endoscopic tools, medical equipment, etc, represent the external parameters. This scenario is an example for an RT system, where our approach can be applied. Here, we define a task to be a surgical action which is set to handle a specific surgical state characterized by a primary range of internal parameters. In this sense, the task consists of the required positioning and movement actions of the robotic arms, instruments and tools. The primary range is the range of parameters that define an initial status of the patient. A task update is a resulting task defined to handle a specific contingency of a surgical state characterized by a range of internal parameters different from the primary range of the original task.

In the next section, we describe our solution. First, we introduce the concept of RT cells and their properties. Afterwards, we list the steps of the algorithm, and proof of boundedness for each step.

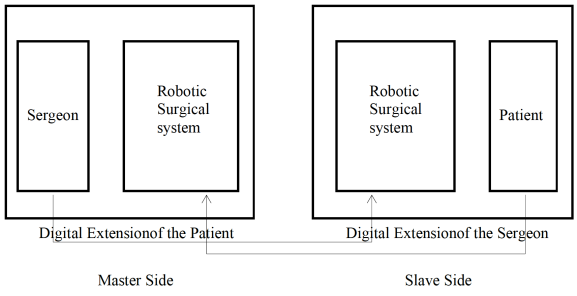


Figure 2. Telerobotic Surgery. [2]

V. SOLUTION

We assume a RT system, which consists of periodic and aperiodic tasks. The scheduling technique to be applied is EDF with relative deadline equal to the period. The aperiodic tasks are served by TBS. All tasks are augmented by additional properties, enabling them to be adapted online. Such an augmented task is denoted by the term "Cell".

We define the $Hyperperiod$ to be an amount of time, that is initially calculated according to the periodic and aperiodic load in the system. The $Hyperperiod$ guarantees a point of time, at which all periodic instances can start their execution. This point of time is the end of current hyperperiod and the beginning of next hyperperiod. The $Hyperperiod$ might change each time the adaptation algorithm takes place. NHP is the point of time at which a hyperperiod ends.

We assume that an adaptation can take place only once per hyperperiod and becomes effective not earlier than the next hyperperiod. The adaptation algorithm is executed by a periodic task with a period equal to the $Hyperperiod$. We define two types of RT cells, the controlling RT cells, and the controlled RT cells. The first one should be able to change the structure and behaviour of the second one. In our approach, we define the EC as the only controlling RT cell in the system. It exists before the system starts. Any other cell in the system is a controlled RT Cell, and abbreviated as RTC. EC becomes an Active Engine-Cell (AEC) once it is activated. An RTC becomes an Active RTC (ARTC) when it is accepted for execution. Each cell inherits the characteristics of RT tasks, and has an additional set of properties. This set enables the organic behaviour to be applied.

A. EC

EC is a periodic cell with period initially calculated as under paragraph 4 in the properties of EC. In the following, we list the properties of the EC:

1) $EC-ID$: the ID of the EC. Each cell in the system has a unique ID.
2) $WorstCaseExecutionTime$ ($WCET_{EC}$): is the worst-case execution time of the AEC.
3) $WorstCasePeriod$ (WCT_{EC}): is the worst-case period of the AEC.
4) $Hyperperiod$: The initial hyperperiod is calculated as the initial NHP (see the calculation of initial NHP on page 7).
5) $NumOfPARTCs$: is the number of periodic ARTCs in the system.
6) $NumOfAARTCs$: is the number of aperiodic ARTCs in the system.
7) $Cost$: is the cost that AEC is assumed to consume. The cost is seen as a function of quality factors.
8) $Active$: is a Boolean variable. It is set to true when the system starts. Whenever the system stops executing, EC becomes not active, and the variable is set to false.

9) $RTCArray$: is the data structure that holds the different cells that exist on the local node and their variants. New cells can be added to the $RTCArray$ at run time. Also, current cells can be updated. Each column is called an $RTClass$. Each $RTClass$ holds a number of variants, which are RTCs dedicated to fulfil the same principal functionality, with different cost and time characteristics. All periodic variants, which belong to the same class, have the same period. The upper bounds of $RTCArray$ dimensions may change online, according to system resources.

B. RTC

An RTC has the following properties:

1) $RTClassID/VariantID$: is a unique ID that differentiates an RTC from other RTCs in the system. Here, $RTClassID$ is the ID of a class of RTCs. In the RTCArray, a different $RTClassID$ is assigned to each column. The $VariantID$ differentiates the different RTCs in the same class (column).
2) $VariantsAllowed$: is a Boolean property that expresses if an RTC is mandatory or not when it should be tested for acceptance by the system. When it is equal to true, all variants that belong to the class of respective RTC should be examined to select the most appropriate variant in the adaptation algorithm. If the property, however, is equal to false, the RTC is considered mandatory to be processed by the adaptation algorithm without considering additional variants of its class.
3) $UpdatingPoints(UP)$: is a set of points in the code of the RTC routine. At these points, the RTC can be substituted by another variant from the RTCArray. The substitution has no influence on the functionality of the RTC. All variants, which have the same $RTClassID$, have a set of updating points with the same number of points, where each point in a specific set has a counterpart point in all the other considered sets. In case of periodic cells, we make a restriction to natural updating points, i.e., the release time of the next instance [4]. Aperiodic RTCs may have a sequence of updating points. The first updating point of an aperiodic cell is its arrival time. The end of the execution of an RTC does not represent an updating point. An x_{th} updating point is represented as $UP[x, y] : x \in \{0, 1, 2, ..\}$, y is the computation time between the starting point of the task and the updating point. When introducing the concept of updating points, we assume that an updating point always has a context switch operation. In this context switch, the AEC has to replace the address of the old variant to the address of the respective location of the new variant. In case of aperiodic variants, we assume the current aperiodic variant just disappears after execution if not explicitly reactivated at some later time. Under this assumption the old variant of the aperiodic RTC just disappears automatically and a potential reactivation automatically relates to the new variant.
4) $ET_{executed}$: is the time that has been spent in executing an aperiodic RTC before starting the current hyperperiod. We assume that this value is always provided by the underlaying RT operating system (RTOS). $ET_{executed}$ is set initially to 0.
5) $NextUpdatingPoint$: a variable that saves the next updating point, which has not been yet reached by the executed code of the RTC.
6) $Triggered$: is a Boolean property that reflects the status of an RTC. If it is equal to true, this means that the RTC is triggered for execution (selected to construct an insertion or deletion request). Otherwise, it is not triggered. Whenever a decision is taken about an RTC to be accepted or not, this property turns to be false. In this case, we assume that the property is turned to false by the AEC.
7) $TriggeringTime$: is the time, at which an RTC is triggered (chosen from the $RTCArray$ to construct a request). Here, we differentiate the arrival time from the TriggeringTime, by defining the arrival time as the time, at which the cell becomes ready for execution.
8) $TriggeringRange$: is the range of time, within which the arrival time of an RTC could be set. It starts at the triggering time. TriggeringRange provides flexibility in choosing arrival times of requests. It is used, in case arrival times are not identical with the next point, at which the hyperperiod of periodic cells is completed (NHP). Our goal is always to set the arrival time of periodic requests equal to NHP, because at this point, we assume that all accepted periodic requests are simultaneously activated (i.e., we assume that all phases to be 0). Our goal is also to set the arrival time of aperiodic requests greater or equal to NHP.
9) $Deletion$: a Boolean property, set to true if the request should be deleted. It is set to false, otherwise.
10) $DeletionTime$: is the time, at which the RTC should to be deleted, if its Deletion property is equal to true.
11) $Active$: is a Boolean variable set to true when the cell is accepted for execution.
12) $ImportanceFactor$: is a number, which represents the expected importance of the RTC, regarding its use in the system. The importance increases by increasing the number. All variants that belong to a specific $RTClass$ have the same $ImportanceFactor$. The $ImportanceFactor$ is considered for filtering the $RTCArray$ when a newly deployed RTC adds a new $RTClass$. The filtering process ensures that the upper bound on the number of $RTClasses$ is not exceeded. Only most important $RTClasses$ are kept.
13) $Essential$: is a Boolean property set to true, when the process of the RTC is essential for the system to operate. Implicitly this means that its importance factor is infinitely high.
14) $Cost$: is an abstract concept. It includes a variety of possible constituents, e.g., memory demand or provided quality like precision of computation. For simplicity reasons, we assume that the cost of the various constituents in a system, which is consumed by any cell is represented by one factor "$Cost$". This factor is a function of several system parameters. Each parameter represents a constituent.
15) $StaticParameters$: is a list of static parameters used in calculating the cost of the RTC. Each parameter

has a name, amount, and a weight.

16) $Type$: the type of an RTC could be periodic or aperiodic. All variants, which have the same $RTClassID$ have the same value of property $Type$.
17) $Cost\_Update$: the updated cost, which should be calculated for an RTC, when it replaces another executing RTC.

C. Complexity variables

The algorithm is bounded if each step needs a bounded time. Time complexity depends on a set of variables.

The variables are:

1) h: The upper bound of the number of columns in the $RTCArray$ (number of $RTClasses$).
2) f: The upper bound of the number of RTCs in an $RTClass$.
3) b: The upper bound of the newly deployed RTCs.
4) PN: an upper bound of number of parameters in the system.
5) QB: The upper bound of requests that can be received in each execution of the EC.
6) SC: The upper bound of the dependent cells, which may construct a request.
7) $m1$:The sum of utilization factors (execution time / period) for the periodic load and AEC (assuming that period of AEC is least common multiple of periods of periodic RTCs), approximated to the next integer number.
8) n: The upper bound of the number of updating points in a cell.
9) GRP: The value of the greatest period available among the periods in the $RTCArray$, and the expected period of the EC.
10) $NInd$: Number of individuals in a generation within the genetic algorithm solving the Knapsack problem.

All parameters have a predefined upper bound, which guarantees boundedness of computation time.

D. Adaptation Algorithm

The following terms are used in the algorithm:

- $ExpPARTCs$: is the set of periodic ARTCs excluding deletion requests.
- $ExpAARTCs$: is the set of aperiodic ARTCs excluding deletion requests.

Calculating an initial NHP is carried out either offline or when starting the system. The initial NHP is calculated as follows:

WCT_{EC} is initially set to the least common multiple of periods of periodic cells in the system. Let $sum1$ denote the sum of initial periodic RTCs utilizations. Initial periodic RTCs are RTCs, which initially are in the system and let $lcm\_initial$ denote the least common multiple of the respective periods. Let $Us\_Initial$ denote the server utilization to handle the aperiodic RTCs, which initially are in the system with respect to their deadlines. Then, the utilization, which can be spent for EC can be calculated by:

$WCET_{EC}/(lcm\_initial \times factor) = 1 - Sum1 - Us\_initial$.

By resolving this equation for factor we obtain

$$factor = \lceil (WCET_{EC}/lcm\_initial \times (1 - Sum1 - Us\_initial)) \rceil \quad (1)$$

The initial NHP is equal to $lcm\_initial \times factor$. This value is set as a period of the EC.

Calculating the initial NHP shows a trade-off. A system, which is highly utilized by its "normal" load suffers from low adaptability as only a small part of the processing power can be assigned to the EC. A high utilization consumed by EC may serve more requests but with longer reaction time. The execution time of the EC depends on b and QB. For this reason, setting b and QB by the system administrator plays a role in this trade off. The execution time increases as these parameters increase. The $WCET_{EC}$ of the EC depends on a couple of parameters. The respective function will be presented at the end of this paper. It is assumed that based on this function and an appropriate model of the underlying hardware the resulting $WCET_{EC}$ can be estimated with sufficient precision.

Each time the EC is executed, following steps take place:

Step 1: Gathering and Filtering the newly deployed RTCs:

The first step of the AEC is to collect the newly deployed RTCs. It stores them in a $WorkingRTCArray$ (a copy of $RTCArray$) following a procedure that ensures to keep the upper bound of the $WorkingRTCArray$ dimensions preserved. Newly deployed RTCs enlarge the solution space when applying the adaptation algorithm. Let b be the upper bound of newly deployed RTCs that can arrive at this step. For the purpose of providing predictability we restrict ourselves to fixed upper bounds in both dimensions of the $WorkingRTCArray$. Let us assume that f is the upper bound of the different variants in each class of the $WorkingRTCArray$, and h is the upper bound of the different $RTClasses$ that can be stored in the $WorkingRTCArray$. If the newly imported RTCs may cause exceeding the upper bound of variants in a column, or the upper bound of columns, the EC preserves the upper bounds by applying a filter procedure. In this context, we discuss following different cases:

1) If the upper bounds of influenced dimensions in $WorkingRTCArray$ will not be exceeded, the RTCs can be added to the $WorkingRTCArray$. See Figure 3, Figure 5 and Figure 6.
   If upper bound of classes is exceeded, one possible heuristic for replacing $RTClasses$ is the ImportanceFactor-based approach. The $RTClasses$ that could be chosen to be replaced by the newly arrived ones are the classes that are not essential or activated. We exclude the classes with the smallest $ImportanceFactor$. For example, let us suppose that there exists in $WorkingRTCArray$ 20 $RTClasses$. All $RTClasses$ have an $ImportanceFactor$ equal to 3, except the third $RTClass$. It has an $ImportanceFactor$ equal to 1. The upper bound of

classes is 20. If we have to add a newly deployed $RTClass$ with an $ImportanceFactor$ equal to 5, and the third $RTClass$ is not essential and not activated, then we can replace the third $RTClass$ by the newly arrived one.

If RTC adds a new periodic variant to an existing column and the upper bound of variants in the column is exceeded, then a decision should be taken, which RTC to drop. One option may be to examine the RTC with the highest C_i/T_i. If it does not result in a successful schedulability test together with the AEC, we exclude it. If it results in a successful schedulability test, we exclude the RTC with the highest cost. For example, let us suppose that there exists 90 variants in the column, where the newly deployed RTC should be added. The upper bound of variants in the column is 90. If the utilization needed by the newly deployed RTC is 0.3, and the highest utilization needed by the variants in the column is 0.7. If the utilization needed by the AEC is 0.6, then we exclude the RTC, which needs the utilization of 0.7. If, however, the highest utilization needed by the variants in the column is 0.2, then we exclude the variant with the highest cost among variants in the column and the newly deployed RTC. If the cost of the newly deployed RTC is 15, and highest cost of variants in the column is 20, we exclude a variant that has the cost 20.

In case the newly deployed RTC adds a new aperiodic variant to an existing column, and this will cause exceeding the upper bound, we exclude the RTC that consumes the highest cost. An arbitrary exclusion can also take place.

2) If a newly arrived column should update a specific existing column, and no variant is active in the existing column, we substitute it by the newly arrived one. See Figure 3 and Figure 4. If there is an active variant in the existing column, we add the newly arrived RTClass to a queue to be considered later. This $UpdateQueue$ may be ordered by the arrival times or an arbitrary other criterion. The upper bound of its capacity is equal to QB. Each element of the $UpdateQueue$ is an array. The array includes the newly deployed $RTClass$, in case this $RTClass$ does not have dependencies. In case a set of dependent $RTClasses$ arrives, the array includes more than one column, the newly deployed $RTClass$ and the other $RTClasses$ in the dependency graph.

Boundedness proof:

- Transmitting the newly deployed RTCs is bounded by b and time for transmission. As we assume a deterministic communication channel between the remote node where newly deployed RTCs reside and the local one, the transmission time is bounded. - The EC applies a filter procedure to preserve upper bounds of the $WorkingRTCArray$. The filter procedure is bounded by one of the $WorkingRTCArray$ dimensions. The formal proof of the boundedness of this step and the next steps is clear when looking at Nassi-Schneiderman diagrams which represent the steps. In [2], time complexity appears on each diagram, and this in turn points out that the step specified by the diagram is done in a bounded time, and the bound depends only on the parameters, which participate in the time complexity formula.

Step 2: Triggering and handling the newly arrived requests:

In the previous step, an $UpdateQueue$ has been constructed, including update requests. In this step, another queue

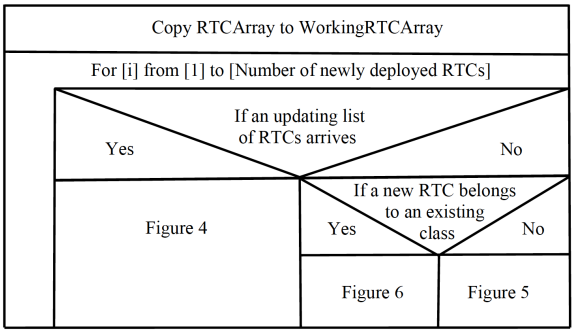


Figure 3. Nassi-Schneiderman Diagram for gathering and filtering the newly deployed RTCs [2].

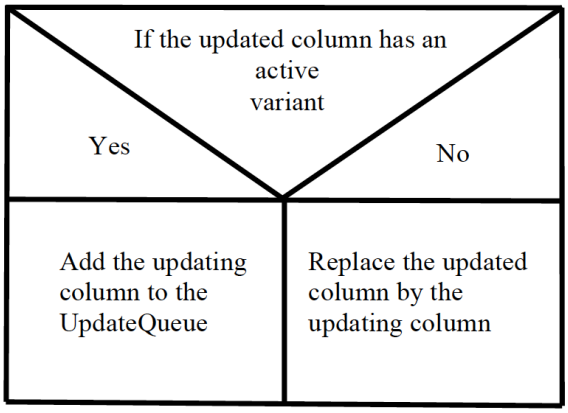


Figure 4. Nassi-Schneiderman Diagram for the arrival of an updating list of RTCs [2].

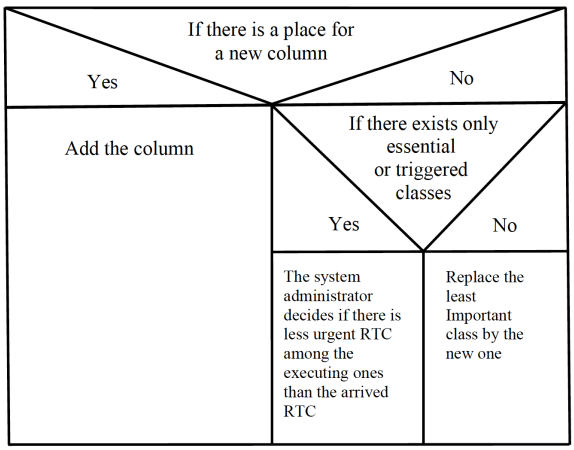


Figure 5. Nassi-Schneiderman Diagram for adding a new column [2].

is constructed. It is called the $TriggeredQueue$, ordered by arrival time or any other criterion. Requests, that are added to this queue, are chosen from the $WorkingRTCArray$. Triggered requests may add or delete RTCs. Triggered requests are chosen according to the necessities of the system. The upper bound of the $TriggeredQueue$ capacity is QB. The EC makes an iteration over items in the $UpdateQueue$ and the $TriggeredQueue$ in parallel. It selects by an arbitrary criterion either an update request or a triggered request. For simplicity, the decision can be made arbitrarily. This selection process is iterated until the number of requests is equal to the upper bound or until no further requests exist. See Figure 7 and Figure 8. Selected requests will be stored in a $RequestQueue$ of bounded size.

If the arrival time of the periodic requests in the $RequestQueue$ is not equal to NHP, their $TriggeringRange$ is examined. If $TriggeringTime \leq$ NHP $\leq TriggeringTime + TriggeringRange$, then the arrival time is set to NHP. Otherwise the requests, which do not satisfy the previous condition, are not accepted and deleted from the $RequestQueue$. After that a notification is sent to the system administrator. For example, if NHP = 10. The arrival time of the request is equal to 9. Its triggeringRange is equal to 5, and the request has been triggered at time 8, we notice that the arrival time of the request is not equal to NHP. For this reason, we examine if $TriggeringTime \leq$ NHP $\leq TriggeringTime + TriggeringRange$. We notice that 8 $\leq$ 10 $\leq$ 8 + 5, so we set the arrival time to 10. The $DeletionTime$ of periodic requests that have to be deleted is set to next natural updating point. If arrival times of aperiodic requests are greater than NHP, they stay the same. If they are smaller than NHP, we set their arrival times the same way as for periodic requests.

If the request includes a set of dependent cells, we assume that their modified arrival times and deadlines are calculated offline by EDF*. If one of the modified arrival times is smaller than NHP, then a fixed offset is applied to all arrival times and deadlines to keep them greater or equal to NHP. See Figure 9.

An update request is represented by an array of RTCs that constructs the $RTClass$ of the update. Updating a set of dependent cells is done under the same rules as updating a cell.

When requests in the $RequestQueue$ proceed for processing by the AEC, the buffers ($UpdateQueue$, $TriggeredQueue$, and $RequestQueue$) become empty.

Boundedness proof:

- A queue of triggered requests (add/delete requests) is constructed in a time bounded by QB.
- The first and second iteration over $UpdateQueue$ and $TriggeredQueue$ is bounded by QB.
- Setting the arrival time of requests is bounded by a constant time.

Step 3: Calculating the cost of quality factors for the system:

A part or the whole set of local parameters might influence the overall quality of the system. This set of parameters includes both parameters of the underlying computing system and of the RT system under consideration. Parameters of the

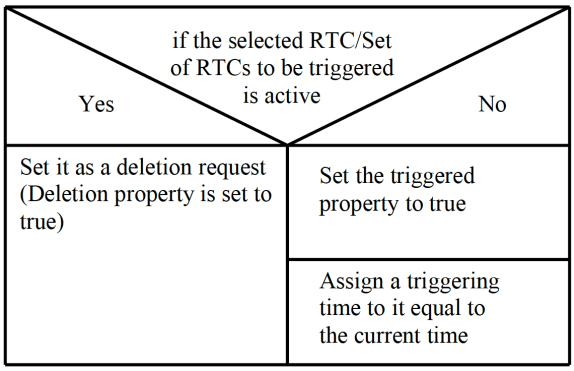


Figure 7. Nassi-Schneiderman Diagram for triggering a request.

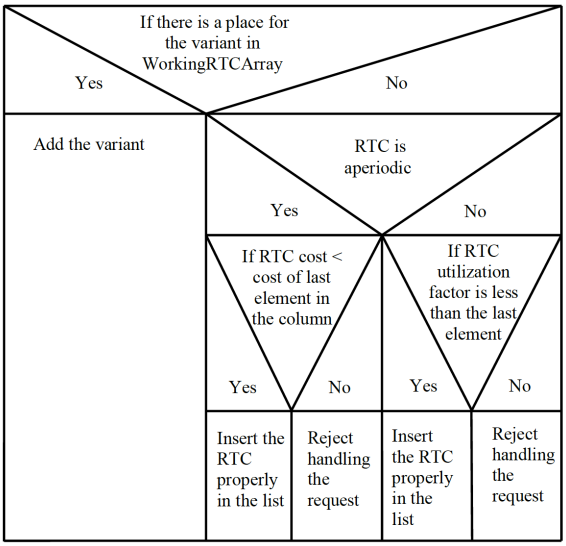


Figure 6. Nassi-Schneiderman Diagram for adding an RTC to an existing column [2].

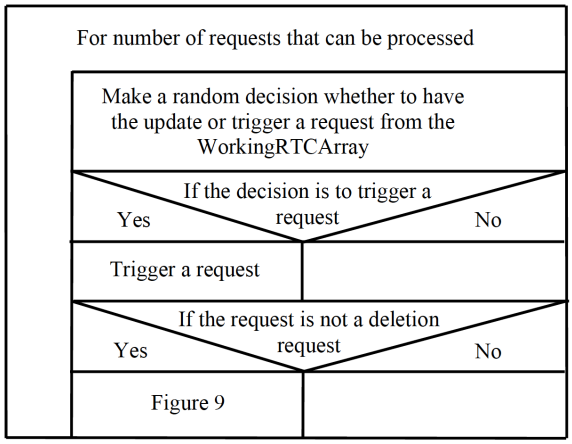


Figure 8. Nassi-Schneiderman Diagram for choosing between an update and a triggered request.

system should be read in each execution of the EC because they might change. This change may affect the result of the adaptation process. E.g., adding new resources may allow accepting a set of requests, which cannot be accepted with less resources. The result of calculating the total cost depending on quality parameters available is called $Cost_{total}$. For example, let us assume that in a remote surgical system, only the following set of parameters are considered: number of cameras, number of robotic arms and number of endoscopic tools. Let us assume that each of these parameters has a weight. Number of cameras is equal to 3. Number of robotic arms is equal to 10. Number of endoscopic tools is equal to 8. The weight of the first parameter is 1. The weight of the second parameter is 4. The weight of the third parameter is 2. Let us assume that $Cost_{total}$ is given by the following function: t $Cost_{total}$ = first parameter $\times$ weight of first parameter + second parameter $\times$ weight of second parameter + third parameter $\times$ weight of third parameter = 3 $\times$ 1 + 10 $\times$ 4 + 8 $\times$ 2 = 3 + 40 + 16 = 59.

Boundedness proof: Under the assumption that an arithmetic operation can be carried out in bounded time, and number of system parameters is bounded by PN, the entire calculation can be carried out in bounded time.

Step 4: Adaptation algorithm:

In this step, we calculate the lowest cost feasible solution over the entire set of $RTClasses$ stored in $AdaptationRTCArray$. The $AdaptationRTCArray$ is constructed as follows:

Constructing AdaptationRTCArray:

1) We copy the variants of the $WorkingRTCArray$ into a temporary array $AdaptationRTCArray$. The $WorkingRTCArray$ is essential for enabling a transaction concept. If the adaptation process turns out to be successful, the $WorkingRTCArray$ will replace the $RTCArray$. If the adaption fails, the $WorkingRTCArray$ will be neglected and the system returns to the previous version.

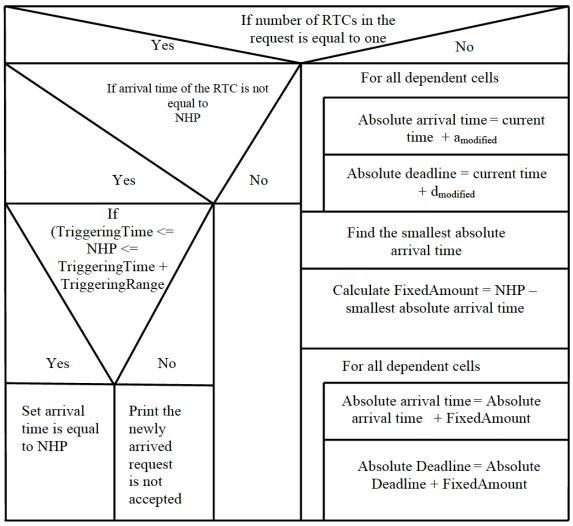


Figure 9. Nassi-Schneiderman Diagram for setting time characteristics of triggered requests [2].

2) We reduce $AdaptationRTCArray$ to contain only the variants, which $RTClassID$ exists in the $ExpPARTCs$ and $ExpAARTCs$ with absolute deadlines exceeding NHP. For each $ExpPARTC$ or $ExpAARTC$, which has the property $VariantsAllowed$ set to false, we do not consider variants that hold the same $RTClassID$ in $AdaptationRTCArray$, other than the ARTC itself.
3) For each aperiodic ARTC that should be deleted, and has absolute deadline exceeding NHP, we add a column including the ARTC as the only variant. If a next possible updating point exists, its execution time is set to $y_{UpdatingPoint}$. The reason is that updating points are the most suitable points to apply deletion, as partial results are delivered on these points. Deleting a cell suddenly on an arbitrary point may cause errors.
4) We then add a column that includes the AEC.
5) We also add the newly triggered requests. If their properties $VariantsAllowed$ are set to true, we add columns that represent $RTClasses$ of the newly triggered variants. If, however, their properties $VariantsAllowed$ are set to false, we add only columns containing the newly triggered variants (a column for each RTC). The value of $VariantsAllowed$ might be different among the different requests.
6) In case there is an update request for an RTC: Adding an aperiodic update is done (only if there exists an updating point after NHP in the aperiodic variant that is running) by adding the updating $RTClass$ that includes the triggered updating variant. In the following, we summarize how to check the existence of an updating point after NHP (only in this case, the updated variant should be excluded when constructing $ExpAARTCs$), and how to set the time characteristics for the variants in the updating column:

First: Determining the set of ARTCs that can be updated: First, we check whether in the current hyperperiod there is capacity left to execute aperiodic ARTCs with deadlines that exceed NHP. This capacity is called "$Amount$".

$$Amount = Hyperperiod - [(\sum_{i=1}^{NumofPARTCS}((Hyperperiod/T_i) \times C_i)) + WCET_{EC} + \sum_{i=1}^{NumofAARTCS-NumOfANHP}(C_i - ET_{executedi})]$$

$NumOfANHP$: refers to the number of aperiodic ARTCs with deadlines that exceed NHP. Based on the value of $Amount$ we now can identify those ARTCs, which definitely result in a completion time later than the current hyperperiod and having an update point after NHP.
$Amount1 = Amount$.

/* We construct a vector of the running aperiodic ARTCs, which deadlines exceed NHP. In the following loop i indicates the i_{th} item in the

vector.*/

If ($Amount1 > 0$) then {
For (i = 1 to $NumofANHP$) {
If ($C_i - ET_{executedi} \leq Amount1$) then {
$Amount1 = Amount1 - (C_i - ET_{executedi})$
}
else {
$C_{i,new} = (C_i - ET_{executedi}) - Amount1$.
If there exist an updating point in $C_{i,new}$, add this ARTC to the set of variants that can be updated
}
}
}
else {
For (i = 1 to $NumofANHP$) {
$C_{i,new} = C_i - ET_{executedi}$
If there exists an updating point in $C_{i,new}$, add this ARTC to the set of variants that can be updated
}
}

Second: Calculation of time characteristics for the updates: If the found updating point is $UP[x, y]$, the arrival time of the j_{th} variant in the updating $RTClass$ is set to the arrival time of the updated variant. The execution time for the j_{th} variant is set to (y + C_j - $\grave{y}$), where $\grave{y}$ is the relative updating point time for the counterpart updating point. The specified absolute deadline for the j_{th} variant is set to max[((D_j - $\grave{y}$) + (Arrival time of the running variant + y)), Absolute Deadline of the running variant that should be updated]

7) Adding a periodic update is done by adding the arrived $RTClass$, which includes the triggered updating variant to $AdaptationRTCArray$. If $VariantsAllowed$ is equal to false, only the triggered updating variant should exist in the column. Otherwise, all variants, which belong to the update exist in the column. The updated variant has to be excluded when constructing $ExpPARTCs$, because executions of periodic instances are completed in each hyperperiod. This means, when a periodic update is applied in the next hyperperiod, no execution of the updated variant can take place.
8) In case there is an update request for a set of aperiodic dependent RTCs, modified arrival times and deadlines are calculated offline. When calculating time characteristics of the variants in the columns that are supposed to update dependent variants, same rules of updating one variant are applied.

In this way, we can use the reduced array in the next step for the adaptation algorithm as each column represents a participant in the selection process. The columns in the array are reordered, so that periodic columns comes first, then AEC, and finally aperiodic columns.

Let us assume that:
the number of columns in $AdaptationRTCArray = Num$.

$\grave{N}$ is the number of columns, which represent the newly triggered aperiodic requests. They are placed as last columns in $AdaptationRTCArray$.

If ($NumOfANHP > 0$) then {

/*In the following, we calculate arrival times, execution times, and $Cost-Update$ for the running aperiodic ARTCs that are stored in $AdaptationRTCArray$, with deadlines exceeding NHP.*/

If ($Amount > 0$) then {

/* Amount as calculated under part "First" is the time left in the current hyperperiod, after excluding the time that should be spent in executing the periodic ARTCs, and aperiodic ARTCs, which deadlines that do not exceed NHP.*/

/*We construct a vector of the running aperiodic ARTCs, which deadlines exceed NHP. We order the elements of this vector according to the increasing absolute deadlines. In the following loop i indicates the i_{th} item in the constructed vector.*/

For (i = 1 to $NumOfANHP$){
If ($C_i - ET_{executedi} \leq Amount$) then {
$Amount = Amount$ - ($C_i - ET_{executedi}$)
Exclude the column of the ith aperiodic variant from $AdaptationRTCArray$. Decrease Num by 1.
}
else{
$C_{i,new} = (C_i - ET_{executedi}) - Amount$.
Set execution time of the variant in $AdaptationRTCArray$ that is equal to the i_{th} variant to $C_{i,new}$.
Set the arrival time of the variant in $AdaptationRTCArray$ that is equal to the i_{th} variant to Arrival time = NHP, if Arrival time < NHP
$Amount = 0$.
}
}
}
else{
For (i = 1 to $NumofANHP$){
$C_{i,new} = C_i - ET_{executedi}$
Set the execution time of the variant in $AdaptationRTCArray$ that is equal to the i_{th} variant to $C_{i,new}$.
Set the arrival time of the variant in $AdaptationRTCArray$ that is equal to the i_{th} variant to Arrival time = NHP, if Arrival time < NHP
}
}
$Cost - Update$ = the cost of the RTC

/*The following iteration is done over the aperiodic RTCs in $AdaptationRTCArray$, which do not belong to

the newly triggered requests.*/

For (k=Number of periodic columns+1..Num-$\grave{N}$){

If ($VariantsAllowed$ = true) && (there exists an updating point in the part of the running variant dedicated for $C_{i,new}$ (after NHP)) then{

/*The following calculations are done to include the possible alternatives for the active aperiodic cells in the knapsack problem.*/

/*A new arrival time, execution time, cost and absolute deadline are calculated for the variants of the k_{th} column in $AdaptationRTCArray$, excluding the running variant in the k_{th} column.*/

Arrival time = Arrival time of the active variant in the k_{th} column.

New execution time is assigned to each variant in the k_{th} Column, excluding the running variant:

$$\grave{C}_{k,new} = (\grave{C} - \grave{y}) + (C_{k,new} - (C_{running,variant} - y))$$

$\grave{C}$ is the execution time of the j_{th} variant, for which we are calculating the attributes, in the k_{th} column.

$C_{k,new}$ is the calculated execution time of the running variant in the k_{th} column.

$C_{running,variant}$: is the original execution time of the running variant in the k_{th} column.

y is the relative updating point time of the next updating point in the running variant.

$\grave{y}$ is the relative updating point time of the counterpart updating point in the j_{th} variant.

$Cost - Update_j$ = Maximum of (Cost of the running variant, cost of the j_{th} variant).

Specified absolute deadline = max (Specified absolute deadline for the running variant, NHP + ($C_{k,new}$ − ($C_{running,variant} - y$)) + specified relative deadline).

}

else We choose the running variant in the k_{th} column.

}

}

Up to know we have prepared the current ecosystem of RTCs, within which we have to find a valid solution with minimized overall cost by making a proper selection of variants.

To find the solution, we solve the following multiple-choice multidimensional knapsack problem:

/*The fist constraint of the knapsack guarantees minimizing the cost of the solution. The second constraint of the knapsack guarantees the schedulability of periodic and aperiodic cells with hard deadlines.*/

$\max \sum_{i=1}^{Num} \sum_{j=1}^{n_i} -Cost_{ij} x_{ij}$

Subject to: $\sum_{i=1}^{Num} \sum_{j=1}^{n_i} W_{ij}^k x_{ij} \leq R^k$

Where: $\sum_{j=1}^{n_i} x_{ij} = 1; i = 1..m$ & $x_{ij} \in \{0,1\}; i = 1..m$ and $j = 1..n_i$, k = 1:3

$W_{ij}^1 = Factor_1/Factor_2$

For any of the periodic RTCs: $Factor1 = C_{ij}$, $Factor2 = T_{ij}$

For the AEC, $Factor1 = WCET_{EC}$, $Factor2 = WCT_{ECtemp}$

For any of the aperiodic RTCs: $Factor1 = 0$, $Factor2 = 1$

WCT_{ECtemp} is calculated as follows:

The expected hyperperiod is calculated as the least common multiple of periods of periodic $ExpPARTCs$ in $AdaptationRTCArray$, and periods of the newly triggered periodic requests in $AdaptationRTCArray$. The resulting value is set as initial value for the expected period of AEC. If the resulting utilization of the RTCs is below 1 then, we examine the total utilization (AEC and RTCs). If it is smaller or equal to 1, we have found the shortest possible expected period for AEC, which at the same time by definition is the hyperperiod. If the total utilization is beyond 1 then the expected hyperperiod has to be extended by a harmonic multiple until the total utilization is no longer beyond 1. If the resulting utilization of the RTCs is 1, the set of chosen RTCs results in a non-feasible solution. In each hyperperiod, only one execution of the AEC is assumed. For this reason, we finally update WCT_{ECtemp}, the expected period of the AEC, to be equal to the expected hyperperiod.

W_{ij}^2 is calculated here in a way different from [1]. In [1], W_{ij}^2 is negative if there is an aperiodic lateness. The sum of weights as stated in the knapsack problem is defined as the aperiodic lateness. If this lateness is smaller than zero (the related knapsack constraint succeeds), the aperiodic lateness indicates a deviation from optimal case of meeting hard deadlines of aperiodic RTCs.

In this paper, we set W_{ij}^2 to be zero if hard deadlines are met ($d_{Calculated,ij} \leq d_{Specified,ij}$). Otherwise it will be equal to a positive value expressing the aperiodic lateness. The sum of weights as stated in the knapsack problem is defined as the aperiodic lateness. If the related knapsack constraint succeeds, hard deadlines of aperiodic RTCs are

met.

$W_{ij}^2 = Factor1 - Factor2$ if $Factor1 > Factor2$, otherwise $W_{ij}^2 = 0$.

For any of the periodic RTCs and the AEC: $Factor1 = 0$, $Factor2 = 0$.

For any of the aperiodic RTCs:

$Factor1 = d_{Calculated,ij}$, $Factor2 = d_{Specified,ij}$.

Where:

$d_{Specified,ij}$: The specified absolute deadline for any aperiodic variant, which belongs to an aperiodic variant in $AdaptationRTCArray$. It is equal to its arrival time + relative deadline of the variant.

$d_{Calculated,ij} = \max\{d_{Calculated(i-1)j_{i-1}}, ArrivalTime_{ij}\} + C_{ij,new}/U_s$.

$d_{Calculated(lp_l)} = 0$.

Where;

$p = j$

l = Number of periodic columns in $AdaptationRTCArray$ +1

$Us = 1 - Up$.

Depending on the different kinds of RTCs to be considered in solving the Knapsack problem, W_{ij}^3 is defined as follows:

$W_{ij}^3 = Cost$ for periodic RTCs stored in $AdaptationRTCArray$

$W_{ij}^3 = Cost - Update$ for running aperiodic RTCs that are stored in $AdaptationRTCArray$

$W_{ij}^3 = Cost$ for added aperiodic RTCs stored in $AdaptationRTCArray$

$R^1 = 1$.

$R^2 = 0$.

$R^3 = Cost_{total}$.

The limit $Cost_{total}$ is optional. If it is set to infinity, then the optimization process tries just to find the lowest cost solution. If the limit is set to a finite value, then the solution space is further limited. If a solution is found, the newly arrived requests are accepted.

The algorithm we are applying to solve the knapsack problem is a genetic algorithm. See subsection E for further details.

If the newly arrived requests are accepted by the system, the ARTCs set or subset, which is represented in $AdaptationRTCArray$ is substituted by the chosen alternatives. Replacing a periodic ARTC means deleting the periodic ones that should be substituted and loading the periodic alternatives at NHP. Replacing an aperiodic ARTC means, the replaced RTC can be treated as a deletion request. When the deletion takes place, the information necessary for replacing the ARTC (transferred from replaced RTC to the replacing one) should be stored. The chosen alternatives are stored in a ready queue. At the NHP, the Active property of the alternatives and for the newly triggered requests is set to true. The Active property of the alternated cells is set to false once they are replaced (deleted). In the aperiodic case, the part of the updated cell that follows the first updating point after NHP is to be replaced. At the replacement point for aperiodic cells, any data of the altered cells or updated cells that might be necessary for the alternatives or updating variants is stored. The *Active* property becomes true for the alternatives. After that, step 5 is applied.

E. Genetic Algorithm

The algorithm we are applying to solve the knapsack problem is a genetic algorithm. In the algorithm, an individual contains exactly one variant for each column in $AdaptationRTCArray$. And a generation may contain one or more individuals. In total there exist up to f^h individuals. Each of them is a potential solution of the Knapsack problem. In the genetic algorithm, we select smaller subsets of individuals and call them Generations. The lowest cost individual of a generation is a preliminary solution of the Knapsack problem. A generation is constructed from a previous one by applying selection and mutation. This process is iterated until no improvement can be observed or a given time limit is reached. We set the first generation to include at least two individuals. The first one is given by selecting from each periodic RTClass the variant with the lowest respective utilization, and from each aperiodic $RTClass$ the variant with lowest respective execution time. The low utilization of a periodic RTC rises the chance of making the sum of all periodic utilizations smaller or equal to 1. The low execution time of an aperiodic RTC rises the chance that the calculated absolute deadline, which is calculated according to TBS, becomes small. As a result, the chance of meeting the hard deadlines of periodic and aperiodic RTCs becomes higher. The second individual is given by the current selection of variants for all $RTClasses$, which are not affected by the adaptation together with all adaptation requests included in the first individual. The first initial individual allows a simple decision whether a solution exists, as if this individual does not fulfil the constraints then there cannot exist any solution. The reason is that we choose the variants in the first individual in a way that reaches the highest chance of satisfying the schedulability test because the periodic utilization is at lowest amount and the calculated aperiodic deadlines according to TBS are at lowest values. The second initial individual is a promising one in the first generation under the assumption that before adaptation we had an optimized system.

Let us assume that the number of individuals in a generation $\leq$ upper bound of number of variants in a class in the $WorkingRTCArray$. The remaining individuals of the first generation may be chosen by any procedure, e.g., by randomly exchanging the selected variants in the columns.

After that WCT_{ECTemp}, server utilization, and absolute deadlines for aperiodic load are calculated for each individual according to TBS [6]. The individuals of a generation are sorted by increasing total costs. This implies that the first individual of this list, provided that the constraints are satisfied, constitutes the preliminary optimum.

If the knapsack constraint $\sum_{i=1}^{Num} \sum_{j=1}^{n_i} W_{ij}^k x_{ij} \leq R^k$ has no solution for the first generation, even under the assumption of $R3$ = infinite, then the adaptation has to be rejected. Otherwise, if it has a solution for a set of individuals, we choose as an intermediate solution the individual, which minimizes the accumulated cost of the chosen RTCs.

In order to potentially improve the solution with the objective to minimize the accumulated cost, we iterate to choose different generations by applying selection and mutation on the individuals, until we either have no further improvement or we reach our predefined time limit.

As an example we assume that the selection process is done by rejecting all constraint-violating individuals and a certain amount of the worst individuals of a generation and that the mutation process is done by replacing an arbitrary RTC in the remaining individuals by another arbitrary RTC from the same column. Selection also implies that the size of the generations may vary (remains bounded). The improvement of the solution is guaranteed by keeping the fittest individuals in the next generation. Choosing an arbitrary variant when applying the mutation may enhance the solution more than choosing a variant with specific characteristics, because there is no characteristic that can guarantee enhancing the solution. We did not apply recombination in our approach. Applying it is a possible option. However, in this case, we should ensure to keep a specific number of individuals in each generation after the selection process, which may enforce keeping a number of constraint-violating individuals in the next generation. This is necessary for the recombination process to take place, because we should assume to have at least two individuals in the previous generation. Figure 10, Figure 12 and Figure 13 describe the solution. Figure 11 is part of the process in Figure 12.

Boundedness proof:

- Operations Step 4 other than the genetic algorithm are bounded by f, h, or f and h.

- The genetic algorithm is bounded because of the following reasons:

1) The operations dedicated to calculate WCT_{ECTemp}, server utilization, and absolute deadlines for the aperiodic load in each individual are bounded by $NInd$, upper bounds of $RTCArray$ dimensions.

2) We can decide whether there exists a feasible solution or not in bounded time. Feasibility can be decided already based on the first generation.

3) The individuals of a generation are sorted by increasing total costs. Sorting is bounded by $NInd$.

4) The optimization is done in bounded time as well. The reason is that we loop from generation to generation until we either have no further improvement or we reach our predefined time limit. The latter termination condition guarantees boundedness.

Step 5: Activate the accepted requests, and update the AEC:

If the newly triggered requests are accepted, the $Active$ property of their RTCs becomes true. They are put into the ready queue of the underlying RTOS. The AEC schedules the first arrival of each request to be at NHP. This is done by loading the accepted RTCs into the memory (transforming them into ARTCs). The scheduler is responsible for loading the accepted RTCs at NHP. The AEC updates its properties, e.g., WCT_{EC} is set to the temporary value WCT_{ECtemp}.

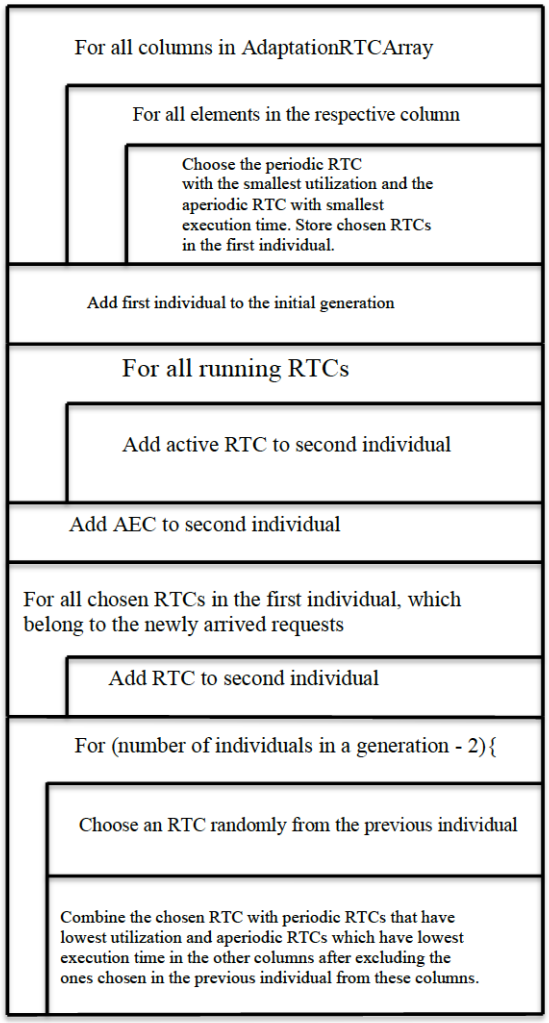


Figure 10. Nassi-Schneiderman Diagram for choosing the initial generation [2].

Hyperperiod = WCT_{EC}.

The AEC updates then its properties according to the changes that will take place. $NumOfAARTCs$ is increased by number of accepted aperiodic RTCs, if the newly triggered RTCs are aperiodic, or the $NumOfPARTCs$ is increased by number of accepted periodic RTCs, if the newly triggered RTCs are periodic. $NumOfAARTCs$ is decreased by number of aperiodic RTCs that are deleted. $NumOfPARTCs$ is decreased by number of periodic RTCs that are deleted. WCT_{EC} is set to the temporary value, which is calculated in step 4 as follows:

$$WCT_{EC} = WCT_{ECtemp}.$$

The hyperperiod is updated according to step 4.

$$Hyperperiod = WCT_{EC}.$$

In case the request is an update for one or several active RTCs, it replaces the $RTClasses$ in the $WorkingRTCArray$, which includes the RTC/RTCs that should be updated by the $RTClass/RTClasses$ of the newly arrived request. We set the $Active$ property of the triggered elements in the newly arrived $RTClasses$ to true. After that, $AdaptationRTCArray$ is set to empty. See Figure 14.

Boundedness proof:

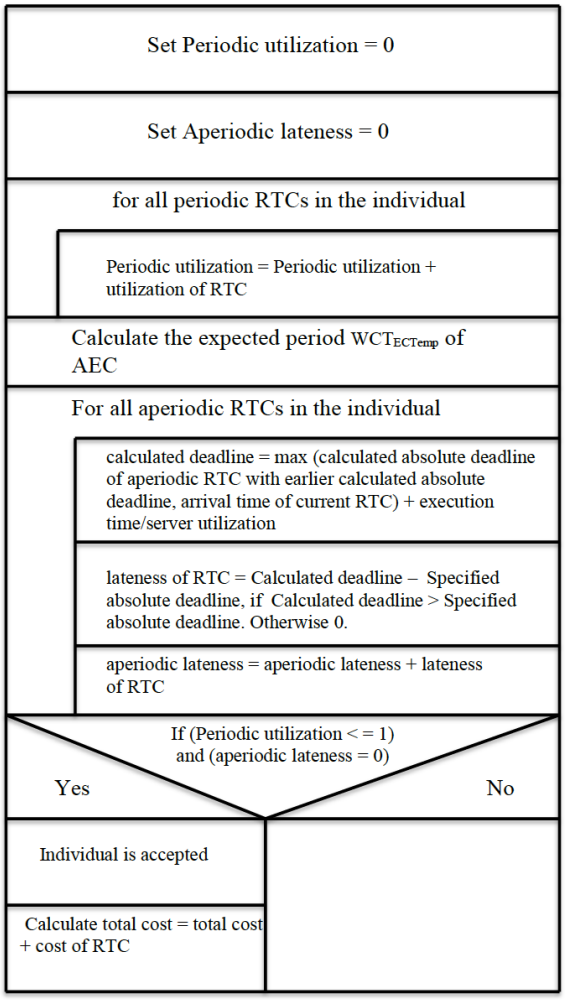


Figure 11. Nassi-Schneiderman Diagram for evaluating an individual [2].

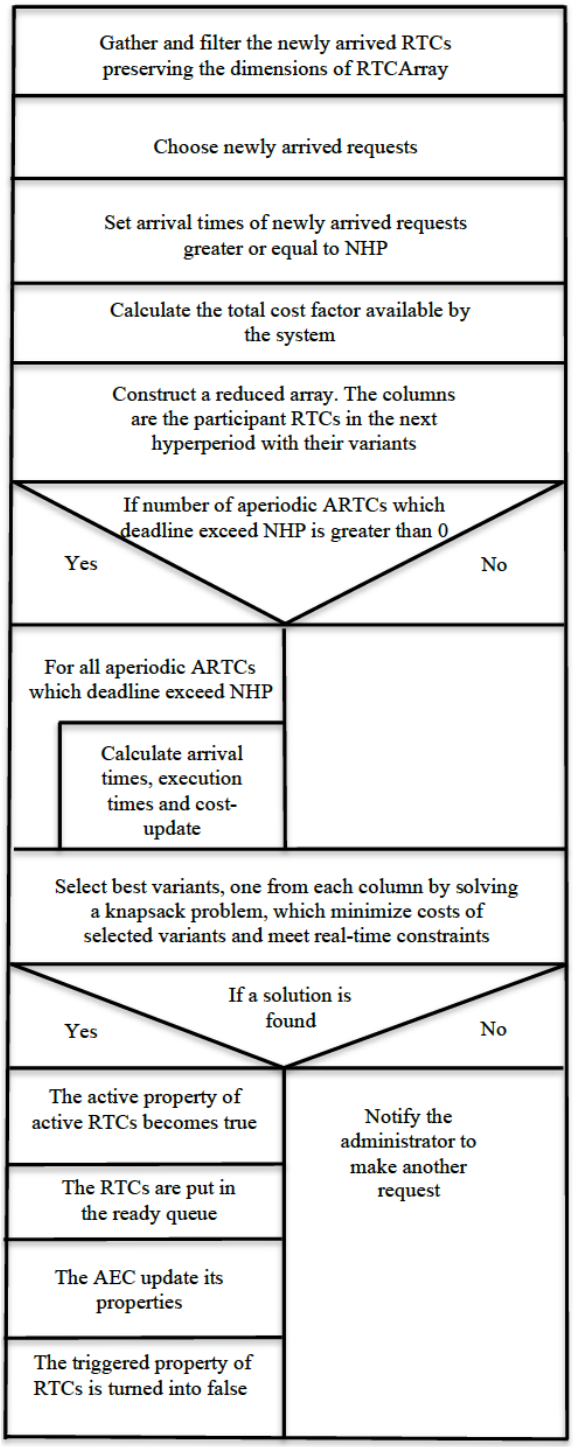


Figure 12. Nassi-Schneiderman Diagram for evaluating a generation [2].

- Activating each accepted request is done in constant time by turning the $Active$ property into true.

- Iterating over newly arrived requests is bounded by QB.

- Updating each of the AEC properties ($NumOfAARTCs$, $NumOfPARTCs$, period of the EC) is also done in constant time.

Step 6: Turning the triggered requests into non-triggered:

The Triggered Property of requests RTCs is turned into false. If the arrived requests are accepted, $WorkingRTCArray$ is copied to $RTCArray$, and then it is set to empty.

Boundedness proof:

- The $Triggered$ property of each request RTC is turned to false in constant time.

- Iterating over the requests in $WorkingRTCArray$ is done in time bounded by h ,f and QB.

- Copying $WorkingRTCArray$ to $RTCArray$ is done in time bounded by f and h. Resetting $WorkingRTCArray$ is done in constant time.

Step 7: Notify the system, in case the requests are not accepted:

If the set of proceeded requests cannot be accepted, then a notification is sent by the AEC to the system for substituting the proceeded set of requests by another set. $Cost_{total}$, WCT_{ECtemp}, The expected hyperperiod, $AdaptationRTCArray$, $ExpPARTCs$ and $ExpAARTCs$ are reset to their initial values. $WorkingRTCArray$ is set to empty.

Boundedness proof:

- A notification is sent to the system administrator in constant time. $Cost_{total}$, WCT_{ECtemp}.

- The expected hyperperiod, $AdaptationRTCArray$,

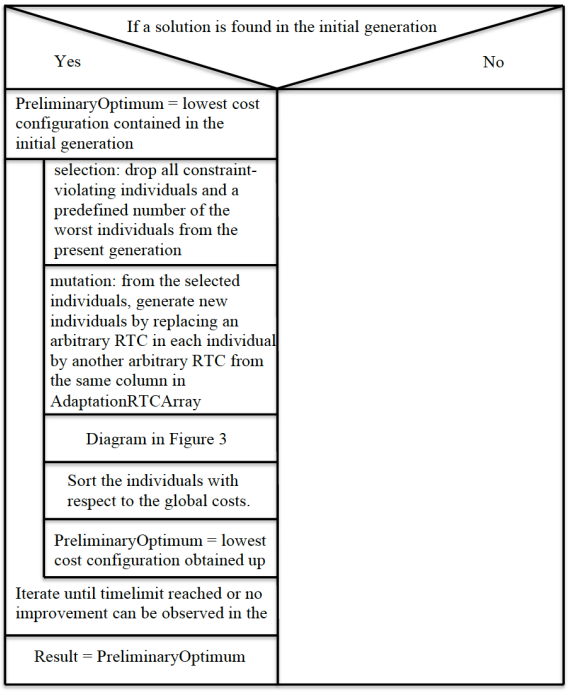


Figure 13. Nassi-Schneiderman Diagram for the genetic algorithm [2].

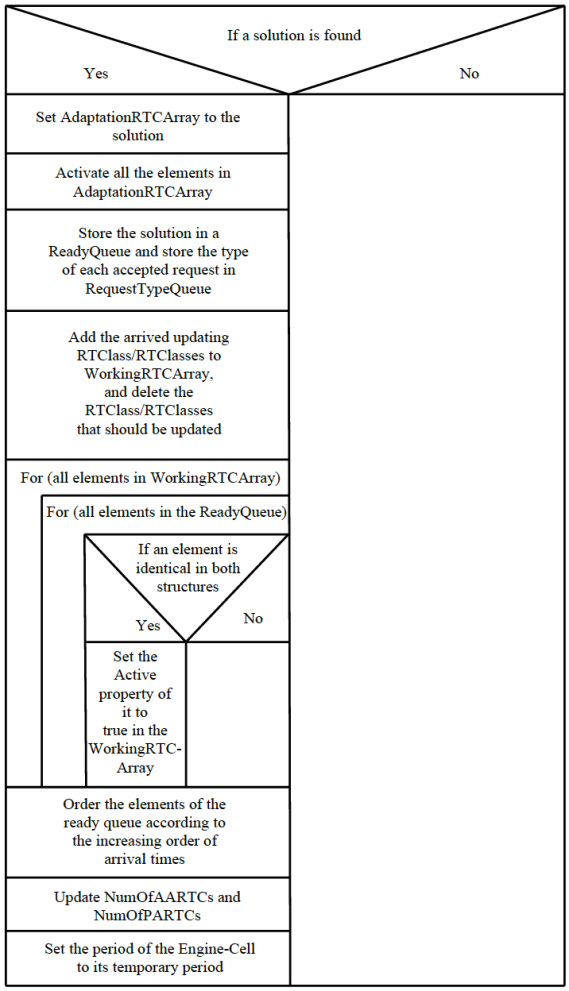


Figure 14. Nassi-Schneiderman Diagram for activating the accepted requests and updating the AEC [2].

```
For i = 1 to N do{      Time consumed in Step 1 is:  Const1 × N
Set x = i;              Time consumed in Step 2  is:  Const2 × N
Set y = 6;              Time consumed in Step 3  is:  Const3 × N
Set z = x + y;          Time consumed in Step 4 is:  Const4 × N
}

Time consumed by the Pseudo code is equal to  Const1 × N + Const2 × N +  Const3 × N + Const4 × N = (Const1 + Const2 + Const3 + Const4) × N.
The time complexity is O(N).
```

Figure 15. Example from extracting time complexity from a pseudo code.

$ExpPARTCs$ and $ExpAARTCs$ are reset in constant time.

- Resetting $WorkingRTCArray$ is done in constant time.

As shown above, the adaptation process takes place in bounded time. Part of this process, however, is the calculation of the $WCET_{EC}$ of the EC. This value depends on a couple of parameters that may vary by each application of the adaption. The following complexity function has been derived in [2] to express the influence of all relevant parameters on time complexity. It is assumed that based on this complexity function and an adequate model of the underlying hardware the resulting $WCET_{EC}$ can be estimated with sufficient precision. The reason is that the complexities, which construct the function can be derived from a pseudo code of the adaptation algorithm. [24] points out how time complexity can be derived from a pseudo code. In [2], the pseudo code was not described, but an estimation was done on Nassi-Schneiderman diagrams. Each diagram points out the step, which is specified by that diagram, by breaking it into smaller steps. One can estimate the complexity of these small steps, as if they were reflecting parts of a pseudo code. In Figure 15, we present an example for extracting the time complexity from a pseudo code.

We find that the algorithm is solved by a quadratic time complexity.

Complexity of the algorithm [2]: Complexity of step 1 + Complexity of step 2 + Complexity of step 3 + Complexity of step 4 + Complexity of step 5 + Complexity of step 6 + Complexity of step 7 = $O(b*h*f)+O(QB*h*f*SC+SC^2)+O(PN)+O(QB*SC*f*h+h^2*f+QB*n+h*n+f^2*h+f*m1+f*logGRP)+O(QB*SC*h*f+h^2*f)+O(h*f*QB*SC)+O(1) = O(b*h*f+PN+f*m1+f*logGRP+f^2*h+h^2*f+QB*n+h*n+h*f*QB*SC+SC^2)$

In Section V, we have first pointed out how to transform the traditional RT tasks into RT cells. For this purpose, we defined the new properties that have to be added to the structure of RT tasks in order to allow executing the tasks as cells. Cells can change their structure an behavior at runtime. In our approach, there is two kinds of cells. EC belongs to one kind, and RTCs belong to the other kind. In this section, we have listed the properties of EC and RTCs. Then we listed all parameters, which may play a role in time complexity of the adaptation algorithm. We defined the parameters. Afterwards, we went through the steps of the algorithm. In each step, we explained how the step is performed. Then, we presented the boundedness proof of the step. Finally, we presented the time complexity of the algorithm.

In the next section, we summarize the content of the paper, and introduce potential future work.

VI. Conclusion and Future Work

In this paper, we provided the details of the algorithmic solution described in [1]. We have showed the proof of boundedness for each step in the algorithm. The solution tries to evolve the system at run time. Each time the EC executes, and new requests exist, there is a possibility to change the RTCs, which construct the system. The EC executes a genetic algorithm, to solve a knapsack problem. The conditions of the problem aim to provide more processor capacity and to minimize the costs by choosing best combination of cells variants. Every individual that results from the genetic algorithm acts as a possible input for the knapsack. The genetic algorithm runs until a best individual is found, or a predefined time limit is reached. The time complexity of the algorithm has been deduced depending on abstract code. Code statements have been modelled by Nassi-Schneiderman diagrams [3]. In [2], time complexities are listed in the diagrams. In the future, we may apply different approaches including different genetic algorithms to solve the knapsack problem. This may provide different optimization output [2]. The problem might also be modelled by means different from the knapsack. Considering communication between cells is an additional aspect that may expand the scope of RT applications, where the algorithm can be applied [2]. The solution is designed for a local node and one remote node, where newly deployed cells can be installed. Later we may design a solution for more than one remote node. Each one is dedicated for a different type or sort of RT cells. By applying this enhancement, we can save costs because nodes can stay where appropriate developers exist [2].

VII. Acknowledgement

This work is based on a PhD thesis done at University of Paderborn, Germany [2].

References

[1] L. Khaluf and F. Rammig, “Organic Self-Adaptable Real-Time Applications,” In the fifteenth international conference on Autonomic and Autonomous Systems (ICAS), pp. 65-71, 2019.

[2] L. Khaluf, “Organic Programming of Dynamic Real-Time Applications,” a PhD thesis, University of Paderborn, 2019.

[3] I. Nassi and B. Schneiderman, “Flowchart Techniques for Structured Programming,” Technical Contributions, Sigplan Notices, pp. 12-26, 1973.

[4] L. Khaluf and F. Rammig, “Organic Programming of Real-Time Operating Systems,” In the ninth international conference on Autonomic and Autonomous Systems (ICAS), pp. 57-60, 2013.

[5] H. Ghetto, M. Silly, and T. Bouchentouf, “Dynamic scheduling of real-time tasks under precedence constraints,” Journal of Real-Time Systems, 2, pp. 181-194, 1990.

[6] M. Spuri and G. C. Buttazzo, “Efficient Aperiodic Service under Earliest Deadline Scheduling,” Real-Time Systems Symposium, pp. 2-11, 1994.

[7] S. Htiouech, S. Bouamama and R. Attia, “OSC: solving the multidimensional multi-choice knapsack problem with tight strategic Oscillation using Surrogate Constraints,” International Journal of Computer Applications (0975 8887), Volume 73 - No. pxc3889883, 2013.

[8] M. M. Akbar, M. S. Rahman, M. Kaykobad, E.G. Manning and G.C. Shoja, “Solving the Multidimensional Multiple-choice Knapsack Problem by constructing convex hulls,” Journal Computers and Operations Research archive, pp. 1259-1273, 2006.

[9] M. Hifi, M. Michrafy and A. Sbihi, “Algorithms for the Multiple-Choice Multi-Dimensional Knapsack Problem,” In: Les Cahiers de la M.S.E : série bleue, Vol. 31, 2003.

[10] Y. Chen and J-K Hoa, “A ”reduce and solve” approach for the multiple-choice multidimensional knapsack problem,” European Journal of Operational Research, pp. 313-322, 2014.

[11] A. Sbihi, M. Mustapha and M. Hifi, “A Reactive Local Search-Based Algorithm for the Multiple-Choice Multi-Dimensional Knapsack Problem,” Computational Optimization and Applications, pp. 271–285, 2006.

[12] Shubhashis K. Shil, A. B. M. Sarowar Sattar, Md. Waselul Haque Sadid, A. B. M. Nasiruzzaman and Md. ShamimAnower, "Solving Multidimensional Multiple Choice Knapsack Problem By Genetic Algorithm & Measuring Its Performance," International Conference on Electronics, Computer and Communication (ICECC), 2008.

[13] A. Duenas, C. D. Martinelly, and G. Tütüncü, "A Multidimensional Multiple-Choice Knapsack Model for Resource Allocation in a Construction Equipment Manufacturer Setting Using an Evolutionary Algorithm," APMS (1), IFIP AICT, Volume 438, pp. 539-546, 2014.

[14] IBM ILOG CPLEX Callable Library version 12.6.2.

[15] D. E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning," 1989.

[16] G. C. Buttazzo, "Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications," Third Edition, Springer, 2011.

[17] H. Rosen, "Handbook of Graph Theory," Series Editor Kenneth, CRC Press, edited by Jonathan L. Gross, Jay Yellen, 2004.

[18] D. Pisinger, "Algorithms for knapsack problems," Dept of Computer Science, University of Kopenhagen, PhD thesis, February, 1995.

[19] F. Streichert, "Introduction to Evolutionary Algorithms," University of Tuebingen, 2002.

[20] http://math.feld.cvut.cz/mt/txtb/3/txe3ba3c.htm. Last visited on 30.05.2020.

[21] https://en.wikipedia.org/wiki/Monotonic˙function. Last visited on 30.05.2020.

[22] http://www.nlreg.com/asymptot.htm. Last visited on 30.05.2020.

[23] J. E. Speich and J. Rosen, "Medical Robotics, Encyclopedia of Biomaterials and Biomedical Engineering," 2004.

[24] https://en.wikipedia.org/wiki/Analysis˙of˙algorithms. Last visited on 31.05.2020.

[25] T. Weise, "Global Optimization Algorithms - Theory and Application", Self-Published, second edition, 2009.

The Collaborative Modularization and Reengineering Approach *CORAL* for Open Source Research Software

Christian Zirkelbach
Software Engineering Group
Kiel University
Kiel, Germany
czi@informatik.uni-kiel.de

Alexander Krause
Software Engineering Group
Kiel University
Kiel, Germany
akr@informatik.uni-kiel.de

Wilhelm Hasselbring
Software Engineering Group
Kiel University
Kiel, Germany
wha@informatik.uni-kiel.de

***Abstract*—Software systems evolve over their lifetime. Changing requirements make it inevitable for developers to modify and extend the underlying code base. Especially in the context of open source software where everybody can contribute, requirements can change over time and new user groups may be addressed. In particular, research software is often not structured with a maintainable and extensible architecture. In combination with obsolescent technologies, this is a challenging task for new developers, especially, when students are involved. In this paper, we report on the modularization process and architecture of our open source research project *ExplorViz* towards a microservice architecture. The new architecture facilitates a collaborative development process for both researchers and students. We explain our employed iterative modularization and reengineering approach *CORAL*, applied measures, and describe how we solved occurring issues and enhanced our development process. Afterwards, we illustrate the application of our modularization approach and present the modernized, extensible software system architecture and highlight the improved collaborative development process. After the first iteration of the process, we present a proof-of-concept implementation featuring several developed extensions in terms of architecture and extensibility. After conducting the second iteration, we achieved a first version of a microservice architecture and an improved development process with room for improvement, especially regarding service decoupling. Finally, as a result of the third iteration, we illustrate our improved implementation and development process representing an entire, separately deployable, microservice architecture.**

Keywords–collaborative software engineering; software modularization; software modernization; open source software; microservices.

I. INTRODUCTION

Software systems are continuously evolving during their lifetime. Changing contexts, legal, or requirement changes such as customer requests make it inevitable for developers to perform modifications of existing software systems. Open source software is based on the open source model, which addresses a decentralized and collaborative software development. In this paper, we report on the iterative modularization process of our open source research project *ExplorViz* towards a more collaboration-oriented development process featuring a microservice architecture based on our previous work [1].

Open research software [2] is available to the public and enables anyone to copy, modify, and redistribute the underlying source code. In this context, where anyone can contribute code or feature requests, requirements can change over time and new user groups may appear. Although this development approach features a lot of collaboration and freedom, the resulting software does not necessarily constitute a maintainable and extensible underlying architecture. Additionally, employed technologies and frameworks can become obsolescent or are not updated anymore. In particular, research software is often not structured with a maintainable and extensible architecture [3]. This causes a challenging task for developers during the development, especially when inexperienced collaborators like students are involved. Based on several drivers, like technical issues or occurring organization problems, many research and industrial projects need to move their applications to other programming languages, frameworks, or even architectures. Currently, a tremendous movement in research and industry constitutes a migration or even modernization towards a microservice architecture, caused by promised benefits like scalability, agility, and reliability [4]. Unfortunately, the process of moving towards a microservice-based architecture is difficult, because there a several challenges to address from both technical and organizational perspectives [5]. We later call the outdated version *ExplorViz Legacy*, and the new version just *ExplorViz*. Our main contributions in this paper are:

- Identification of technical and organizational problems in our monolithic open source research project *ExplorViz*.
- An iterative modularization and reengineering process focusing on collaborative development applied on our project moving towards a microservice architecture in three iterations.
- A proof-of-concept implementation, followed by an evaluation based on several developed extensions, as the result of the first iteration.
- An improved software architecture based on microservices and development process after our second iteration.
- Finally, after our third iteration, an entire and separately deployable microservice architecture.

The remainder of this paper is organized as follows. In Section II, we illustrate our problems and drivers for a

modularization and architectural modernization. Afterwards, we present the initial state our software system and underlying architecture of *ExplorViz Legacy* in Section III. Our employed modularization and modernization process as explained in Section IV. The following first iteration of this process as well as the target architecture of *ExplorViz* are described in Section V. Section VI concludes the first iteration with a proof-of-concept implementation in detail, including an evaluation based on several developed extensions. The second iteration of our process in terms of achieving a first microservice architecture is presented in Section VII. As there was still room for improvement, we describe how we further improved our microservice architecture and development process in Section VIII. Section IX discusses related work on modularization and modernization towards microservice architectures. Finally, the conclusions are drawn, which includes a summary, depicts lessons learned, and gives an outlook for future work.

II. Problem Statement

The open source research project *ExplorViz* started in 2012 as part of a PhD thesis and is further developed and maintained until today. *ExplorViz* enables a live monitoring and visualization of large software landscapes [6], [7]. In particular, the tool offers two types of visualizations – a landscape-level and an application-level perspective. The first provides an overview of a monitored software landscape consisting of several servers, applications, and communication in-between. The second perspective visualizes a single application within the software landscape and reveals its underlying architecture, e.g., the package hierarchy in Java, and shows classes and related communication. The tool has the objective to aid the process of system and program comprehension for developers and operators. We successfully employed the software in several collaboration projects [8], [9] and experiments [10], [11]. The project is developed from the beginning on *Github* with a small set of core developers and many collaborators (more than 40 students) over the time. Several extensions have been implemented since the first version, which enhanced the tool's feature set. Unfortunately, this led to an unstructured architecture due to an unsuitable collaboration and integration process. In combination with technical debt and issues of our employed software framework and underlying architecture, we had to perform a technical and process-oriented modularization. Since 2012, several researchers, student assistants, and a total of 31 student theses as well as multiple projects contributed to *ExplorViz*. We initially chose the Java-based Google Web Toolkit (*GWT*) [12], which seemed to be a good fit in 2012, since Java is the most used language in our lectures. *GWT* provides different wrappers for Hypertext Markup Language (HTML) and compiles a set of Java classes to JavaScript (JS) to enable the execution of applications in web browsers. Employing *GWT* in our project resulted in a monolithic application (hereinafter referred to as *ExplorViz Legacy*), which introduced certain problems over the course of time.

A. Extensibility & Integrability

ExplorViz Legacy's concerns are divided in core logic (core), predefined software visualizations, and extensions. When *ExplorViz Legacy* was developed, students created new Git branches to implement their given task, e.g., a new feature. However, there was no extension mechanism that allowed the integration of features without rupturing the core's code base. Therefore, most students created different, but necessary features in varying classes for the same functionality. Furthermore, completely new technologies were utilized, which introduced new, sometimes even unnecessary (due to the lack of knowledge), dependencies. Eventually, most of the developed features could not be easily integrated into the master branch and thus remained isolated in their created feature branch.

B. Code Quality & Comprehensibility

After a short period of time, modern JS web frameworks became increasingly mature. Therefore, we started to use *GWT*'s JavaScript Native Interface (JSNI) to embed JS functionality in client-related Java methods. For example, this approach allowed us to introduce a more accessible JS-based rendering engine. Unfortunately, JSNI was overused and the result was a partitioning of the code base. Developers were now starting to write Java source code, only to access JS, HTML, and Cascading Style Sheets (CSS). This partitioning reduced the accessibility for new developers. Furthermore, the integration of modern JS libraries in order to improve the user experience in the frontend was problematic. Additionally, Google announced that JSNI would be removed with the upcoming release of Version 3, which required the migration of a majority of client-related code. Google also released a new web development programming language, named *DART*, which seemed to be the unofficial successor of *GWT*. Thus, we identified a potential risk, if we would perform a version update. Eventually, JSNI reduced our code quality. By code quality, we understand the maintainability of the source code, which includes the concepts of analyzability, changeability, and understandability [13]. Our remaining Java classes further suffered from ignoring some of the most common Java conventions and resulting bugs. Students of our university know and use supporting software for code quality, e.g., static analysis tools such as *Checkstyle* [14] or *PMD* [15]. However, we did not define a common code style supported by these tools in *ExplorViz Legacy*. Therefore, a vast amount of extensions required a lot of refactoring, especially when we planned to integrate a feature into the core.

C. Software Configuration & Delivery

In *ExplorViz Legacy*, integrated features were deeply coupled with the core and could not be easily taken out. Often, users did not need all features, but only a certain subset of the overall functionality. Therefore, we introduced new branches with different configurations for several use cases, e.g., a live demo. Afterwards, users could download resulting artifacts, but the maintenance of related branches was cumbersome. Summarized, the stated problems worsened the extensibility, maintainability, and comprehension for developers of our software. Therefore, we were in need of modularizing and modernizing *ExplorViz Legacy*.

III. *ExplorViz Legacy*

In order to understand the modularization process, we provide more detailed information about our old architecture in the following. The overall architecture and the employed

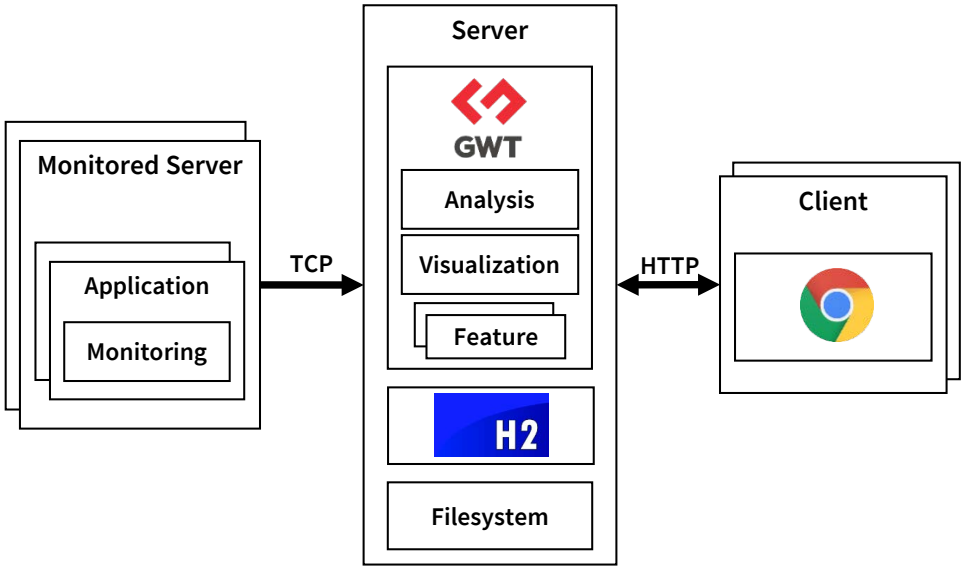


Figure 1: Architectural overview and software stack of the monolithic *ExplorViz Legacy*.

software stack of *ExplorViz Legacy* is shown in Figure 1. We are instrumenting applications, regardless whether they are native applications or deployed artifacts in an application server like *Apache Tomcat*. The instrumentation is realized by our monitoring component. The component employs in the case of Java *AspectJ*, an aspect-oriented programming extension for Java [16]. *AspectJ* allows us to intercept an application by bytecode-weaving. Thereby, we can gather necessary monitoring information for analysis and visualization purposes. Subsequently, this information is transported via Transmission Control Protocol (TCP) towards a server, which hosts our *GWT* application. This part represents the two major components of our architecture, namely *analysis* and *visualization*. The *analysis* component receives the monitoring information and reconstructs traces. These traces are stored in the file system and describe a software landscape consisting of monitored applications and communication in-between. Our user-management employs the *H2* database [17] to store related data. The software landscape *visualization* is provided via Hypertext Transfer Protocol (HTTP) and is accessible by clients with a web browser. *GWT* is an open source framework, which allows to develop JS front-end applications in Java. It facilitates the usage of Java code for server (backend) and client (frontend) logic in a single web project. Client-related components are compiled to respective JS code. The communication between frontend and backend is handled through asynchronous remote procedure calls (ARPC) based on HTTP. The usage of ARPC allows non-professional developers, in our case computer science students, to easily extend our existing open source research project. ARPC enables a simple exchange of Java objects between client and server. In *ExplorViz Legacy*, the advantages of *GWT* proved to be a drawback, because every change affects the whole project due to its single code base. New developed features were hard-wired into the software system. Thus, a feature could not be maintained, extended, or replaced by another component with reasonable effort. This situation was a leading motivation for us to look for an up-to-date framework replacement. We intended to take advantage of this situation and modularize our software system. The plan was to move from a monolithic to a distributed (web) application divided into separately maintainable and deployable backend and frontend components.

Our open source research project is publicly accessible since the beginning on *Github* and is licensed under the *Apache License, Version 2.0*. The development process facilitated the maintainability and extensibility of our software by means of so-called feature branches. Every code change, e.g., a new feature or bugfix, had to be implemented in a separated feature branch based on the master branch. This affected not only the core developers (researchers), but also student assistants, or students during a thesis or project. After performing a validation on the viability and quality of the newly written source code, the branch needed to be merged into the master project and thus permanently into the project. This fact often led to an intricate and time-consuming integration process, since all developers worked on a single code base. For that reason, we had to improve our development process to perform a modularization and technical modernization.

The previously mentioned drawbacks in *ExplorViz Legacy* were our initial trigger for a modularization and modernization. Additionally, recent experience reports in literature were published about successful applications of alternative technologies, e.g., Representational State Transfer (REST or RESTful) Application Programming Interfaces (API) [18], [19]. In the following, we describe our employed, iterative modularization and reengineering approach *CORAL*, which guided us through this process.

IV. The Modularization and Reengineering Approach *CORAL*

Our Collaborative Reengineering and Modularization Approach (*CORAL*) addresses problems regarding the modernization and modularization of open source research projects in technical and organizational aspects. This collaborative,

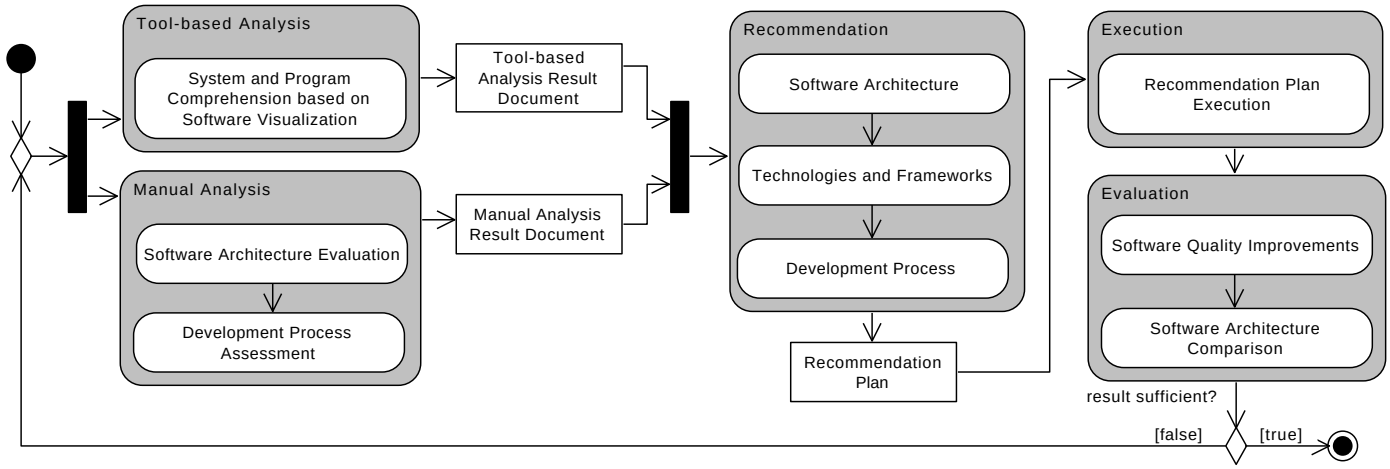


Figure 2: UML activity diagram illustrating our iterative modularization and reengineering approach *CORAL*.

tool-employing approach supports developers and operators in modularizing and modernizing their software systems in an iterative manner. Basically, the approach consists of five, consecutive activities to support the modularization and reengineering of existing software projects and involved systems. Figure 2 gives an overview of the approach in form of an UML activity diagram. The five activities (colored in gray) are *Manual Analysis*, *Tool-based Analysis*, *Recommendation*, *Execution*, and *Evaluation*. In the following, the activities are briefly described.

A. Manual Analysis

An existing software project and involved systems, which are in need of modularization and modernization, have to be analyzed first (by the developers). Therefore, we need to take a look at the underlying architecture, employed technologies, and tools. This task includes a software architecture and modernization evaluation, in order to identify and reassess legacy source code, frameworks and utilized libraries, and execution environments. The software architecture evaluation task is divided into four parts – (i) a software architecture review, (ii) the application of the software architecture evaluation method *ATAM* [20], (iii) the identification of technical debt, and (iv) the examination of employed technologies and frameworks. For guidelines and approaches for evaluating software architectures, we refer to [21]–[23]. Additionally, the developers need to contribute their knowledge of known technical debt, existing documentation, and their current development process. For assessing and evaluating software development processes, we refer to [24], [25]. The results of this activity are summarized in form of a result document.

B. Tool-based Analysis

Afterwards, the system is analyzed with tools, which aid the modularization process by detecting (technical) flaws, possible shortcomings, and optimization potential. In detail, we focus on the aspect of understanding the software system. We address this aspect by employing the software visualization tool *ExplorViz* itself in order to aid the system and program comprehension process. We employ *ExplorViz* to achieve a better understanding of the software systems we want to modularize and modernize within our approach. *ExplorViz* was already successfully utilized for comprehension purposes in several scientific [8], [9] and industrial collaborations. By utilizing *ExplorViz* for the program comprehension process, we take advantage of software visualizations instead of software artifacts like source code or documentation. Thus, we can enhance our previously obtained knowledge about the software systems from discussions and interviews with the software developers. Finally, we document our findings in form of a result document.

C. Recommendation

In this activity, we take a look into the analysis result documents of the *Manual Analysis* and *Tool-based Analysis* activities, and design a recommendation plan in collaboration with the developers. The recommendation plan is based on the results and examines possible (target) architectures, technologies, and frameworks. Thereby, we also take the employed development process into account. The purpose is to facilitate synergy effects between the software system and the corresponding development process. In the best case, we achieve a collaborative development process, which supports the planned modularization and modernization from the beginning.

D. Execution

After discussing the presented options leading towards a recommendation plan in the last activity, we need to prepare the execution of it. More precisely, we work out a proof-of-concept implementation of the recommendation plan first. Thus, we can verify the necessary technical adaptions in general and are able to perform the reengineering and modularization process afterwards on a solid basis.

E. Evaluation

Once we executed our recommendation plan, we need to evaluate its impact on the software system. Therefore, we focus on comparing the software quality based on metrics provided by software quality tools on one hand and the

software architecture through visual comparison on the other hand. Typically, the results of the evaluation are not sufficient after only one execution. Thus, it is likely, that the overall approach needs to be conducted multiple times in order to achieve an acceptable state.

V. First Iteration: Modularization Process and Architecture of *ExplorViz*

Within *ExplorViz Legacy*, we applied the above mentioned process, which guided us through our modularization process from performing a first requirement analysis and defining goals towards our actual state. Summarized, we performed multiple iterations of the process until we reached an entire, maintainable, and especially extensible microservice architecture. In the following, the first iteration of the process is described.

A. Requirement Analysis and Goals

We no longer perceived advantages of preferring *GWT* over other web frameworks. During the modularization planning phase, we started with a requirement analysis for our modernized software system and identified technical and development process related impediments in the project. We kept in mind that our focus was to provide a collaborative development process, which encourages developers to participate in our research project [26]. Furthermore, developers, especially inexperienced ones, tend to have potential biases during the development of software, e.g., they make decisions on their existing knowledge instead of exploring unknown solutions [27].

As a result, we intended to provide plug-in mechanisms for the extension of the backend and frontend with well-defined interfaces. We intended to encourage developers to try out new libraries and technologies, without rupturing existing code. According to [28], the organization of a software system implementation is not an adequate representation of a system's architecture. Thus, architectural changes towards the implementation of a software system have to be documented before or at least shortly after the realization. If this aspect is not addressed, the architecture model has a least to be updated based on the implementation in a timely manner. Thus, we took this into account in order to enhance our development process. Architectural decay in long-living software systems is also an important aspect. Over time, architectural smells manifest themselves into a system's implementation, whether they were introduced into the system from the beginning or later during development [29]. For the modularization process of our software system it was necessary to look for such smells to eliminate them in the new system. In the end, we identified the following goals for our modularization and modernization process:

- The project needs to be stripped down to it's core, anything else is a form of extension.
- We need to focus on the main purpose of our project – the visualization of software landscapes and architectures. Thus, we need to look for a monitoring alternative.
- The backend and frontend should be separately deployable and technologically independent. The latter goal allows us to replace them with little effort. Additionally, they store their own data and use no centralized storage or database.
- Scaffolds or dummy-projects are provided for the development of extensions.
- We stick to the encapsulation principle and provide well-defined interfaces.
- The overall development process needs to be enhanced, e.g, by using Continuous Integration (CI) and quality assurance (QA), like code quality checks.

In general, there exist many drivers and barriers for microservice adoption [30]. Typical barriers and challenges are the required additional governance of distributed, networked systems and the decentralized persistence of data. After we applied the two activities *Manual Analysis* and *Tools-based Analysis* within our iterative *CORAL* approach, we agreed within the *Recommendation* activity to build our recommendation plan upon an architecture based on microservices. This architectural style offers the ability to divide monolithic applications into small, lightweight, and independent services, which are also separately deployable [4], [31]–[33]. However, the obtained benefits of a microservice architecture can bring along some drawbacks, such as increased overall complexity and data consistency issues [34]. Adopting the above mentioned goals lead us finally to the microservice-based architecture shown in Figure 3.

B. Extensibility & Integrability

In a first step, we modularized our *GWT* project into two separated projects, i.e., backend and frontend, which are now two self-contained microservices. Thus, they can be developed technologically independent and deployed on different server nodes. In detail, we employ distinct technology stacks with independent data storage. This allows us to replace the microservices, as long as we take our specified APIs into account. We tried to evaluate how we can facilitate the development for our main collaborators, i.e., our students. Therefore, our selection of technologies was driven by the students' education at the Kiel University. The backend is implemented as a Java-based web service based on *Jersey* [35], which provides a RESTful API via HTTP for clients. We chose Jersey, because of its JAX-RS compliance. In our opinion, Jersey is a mature framework and due to its HTTP roots it is easy to understand for developers, especially collaborators such as students. *Jersey* implements the Servlet 3.0 specification, which offers javax.servlet.annotations to define servlet declarations and mappings. We assume that the usage of the Servlet 3.0 specification eases the development process in the backend, especially for students. Furthermore, we replaced our custom-made monitoring component by the monitoring framework *Kieker* [36]. This framework provides an extensible approach for monitoring and analyzing the runtime behavior of distributed software systems. Monitored information is sent via TCP to our backend, which employs the filesystem and *H2* database for storage. *Kieker* employs a similar monitoring data structure, which fits our replacement requirements perfectly. The frontend uses the JS framework *Ember.js*, which enables us to offer visualizations of software landscapes to clients with

a web browser [37]. *Ember.js* was chosen, since its core idea of using addons to modularize the application, i.e., the frontend of *ExplorViz* is a good practice in general. Furthermore, the software ecosystem of *Ember.js* with community-driven addons is tremendous and their developer team frequently updates the framework with new features. Since *Ember.js* is based on the *model-view-viewmodel* architectural pattern, developers do not need to manually access the Document Object Model and thus need to write less source code. *Ember.js* uses *Node.js* as execution environment and emphasizes the use of components in web sites, i.e., self-contained, reusable, and exchangeable user interface fragments [38]. We build upon these components to encapsulate distinct visualization modes, especially for extensions. Communication, like a request of a software landscape from the backend, is abstracted by so-called *Ember.js* adapters. These adapters make it easy to request or send data by using the *convention-over-configuration* pattern. The introduced microservices, namely backend and frontend, represent the core of *ExplorViz*. As for future extensions, we implemented well-defined extension interfaces for both microservices, that allow their integration into the core.

C. Code Quality & Comprehensibility

New project developers, e.g., students, do not have to understand the complete project from the beginning. They can now extend the core by implementing new mechanics on the basis of a plug-in extension. Extensions can access the core functionality only by a well-defined read-only API, which is implemented by the backend, respectively frontend. This high level of encapsulation and modularization allows us to improve the project, while not breaking extension support. Additionally, we do no longer have a conglomeration between backend and frontend source code, especially the mix of Java and JS, in single components. This eased the development process and thus reduced the number of bugs, which previously occurred in *ExplorViz Legacy*. Another simplification was the use of *json:api* [39] as data exchange format specification between backend and frontend, which introduced a well-defined JavaScript Object Notation (JSON) format with attributes and relations for data objects. This minimizes the amount of data and round trips needed when making API calls. Due to its well-defined structure and relationship handling, developers are greatly supported when exchanging data.

D. Software Configuration & Delivery

One of our goals was the ability to easily replace the microservices. We fulfill this task by employing frameworks, which are exchangeable with respect to their language domain, i.e., Java and JS. We anticipate that substituting these frameworks could be done with reasonable effort, if necessary. Furthermore, we offer pre-configured artifacts of our software for several use cases by employing *Docker* images. Thus, we are able to provide containers for the backend and frontend or special purposes, e.g., a fully functional live demo. Additionally, we implemented the capability to plug-in developed extensions in the backend, by providing a package-scanning mechanism. The mechanism scans a specific folder for compiled extensions and integrates them at runtime.

VI. Proof-of-Concept Implementation

In order to execute and afterwards evaluate the recommendation plan we designed before, we realized a proof-of-concept implementation and split our project as planned into two separate projects – a backend project based on *Jersey*, and a frontend project employing the JS framework *Ember.js*. Both frameworks have a large and active community and offer sufficient documentation, which is important for new developers. As shown in Figure 3, we strive for an easily maintainable, extensible, and plug-in-oriented microservice architecture. Since the end the first iteration of our modularization and modernization process in early 2018, we were able to successfully develop several extensions both for the backend and the frontend. Four of them are described in the following.

A. Application Discovery

Although we employ the monitoring framework *Kieker*, it lacks a user-friendly, automated setup configuration due to its framework characteristics. Thus, users of *ExplorViz* experienced problems with instrumenting their applications for monitoring. In [40], we reported on our application discovery and monitoring management system to circumvent this drawback. The key concept is to utilize a software agent that simplifies the discovery of running applications within operating systems. An example visualization of the extension's user-interface is shown in Figure 4. The figure shows three discovered applications on a monitored server. Furthermore, this extension properly configures and manages the monitoring framework *Kieker*. More precisely, the extension is divided in a frontend extension, providing a configuration interface for the user, and a backend extension, which applies this configuration to the respective software agent lying on a software system. Then, the software agent is able to apply the chosen configuration towards *Kieker* for the application monitoring.

Finally, we were able to conduct a first pilot study to evaluate the usability of our approach with respect to an easy-to-use application monitoring. The improvement regarding the usability of the monitoring procedure of this extension was a great success. Thus, we recommend this extension for every user of *ExplorViz*.

B. Virtual Reality Support

An established way to understand the complexity of a software system is to employ visualizations of software landscapes. However, with the help of visualization alone, exploring an unknown software system is still a potentially challenging and time-consuming task. In the past years, Virtual Reality (VR) techniques emerged at the consumer market. Starting with the Oculus Rift DK1 head-mounted display (HMD), which was available at the end of 2013, the VR devices constituted a major step towards the consumer market. Based on this development, modern VR approaches became affordable and available for various research purposes. A similar development can be observed in the field of gesture-based interfaces, when Microsoft released their Kinect sensor in 2010 [41]. A combination of both techniques offers new visualization and interaction capabilities for newly created software, but can also improve reverse engineering tasks of existing software by means of immersive user experience.

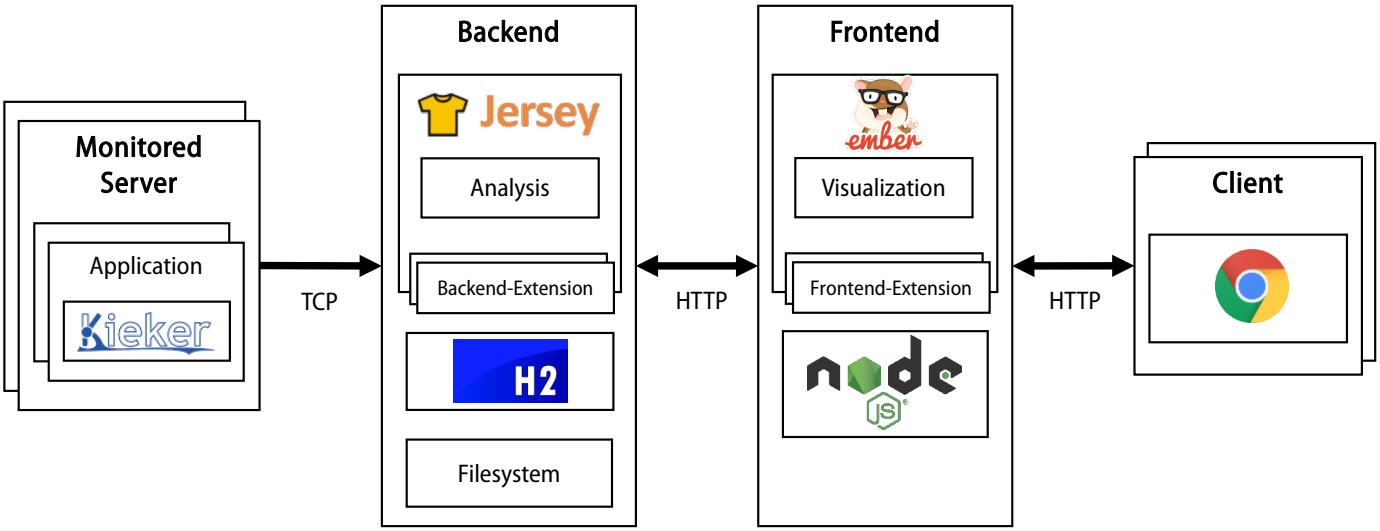


Figure 3: Architectural overview and software stack of the modularized *ExplorViz* (after the first iteration).

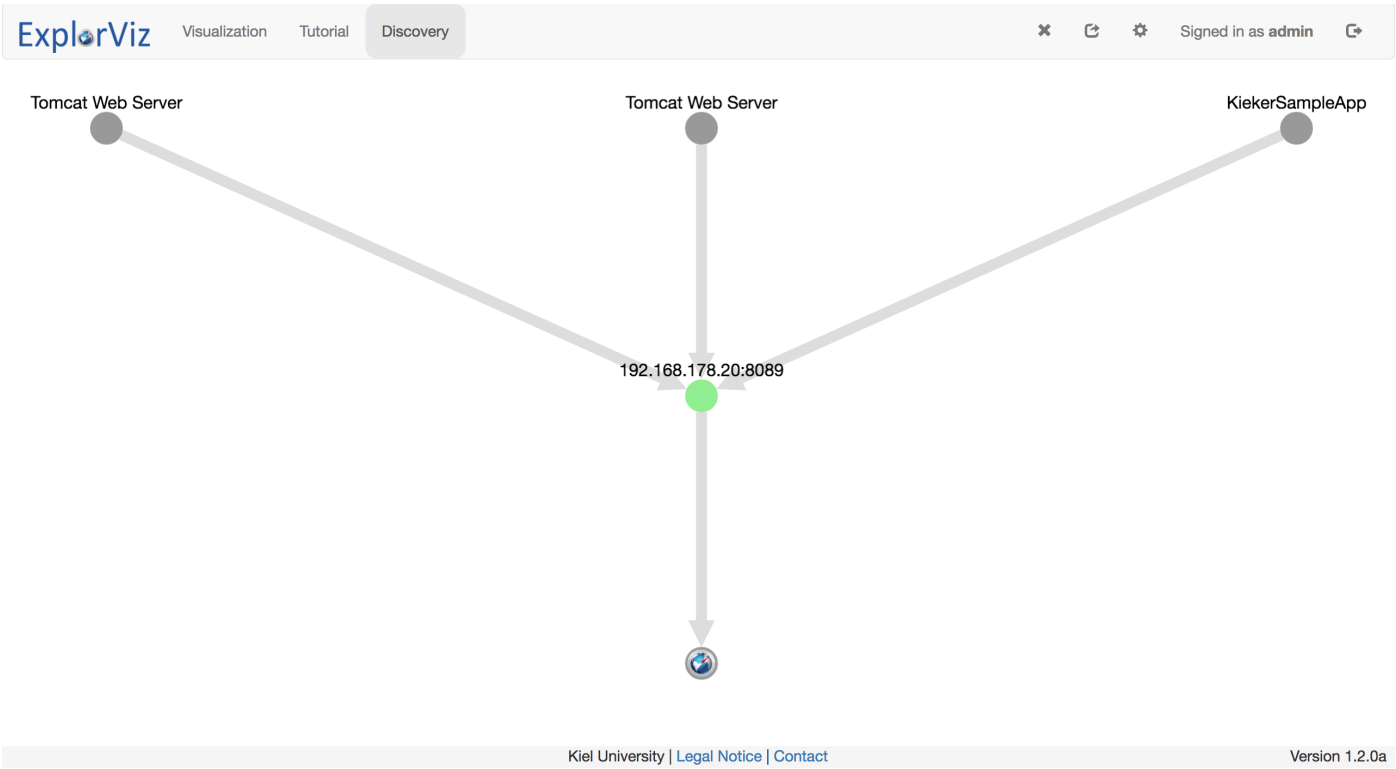


Figure 4: Screenshot of the application discovery extension of *ExplorViz*.

Based on an in-depth 3D visualization and a more natural interaction, compared to a traditional 2D screen and input devices like mouse and keyboard, the user gets a more immersive experience, which benefits the comprehension process [42]. VR can offer an advantage in comparison to existing developer environments to enable new creative opportunities and potentially result in higher productivity, lower learning curves, and increased user satisfaction [43]. For this extension, five students followed a new approach using VR for exploring software landscapes collaboratively based on our previous work [44]. They employed severals HMDs (HTC Vive, HTC Vive Pro, and Oculus Rift) to allow a collaborative exploration and comprehension of software in VR. A screenshot of the VR extension featuring the application-perspective and visualized VR controllers is shown in Figure 5. The collaborative VR approach builds upon our microservice architecture and employs WebSocket connections to exchange data to achieve modular extensibility and high performance for this real-time

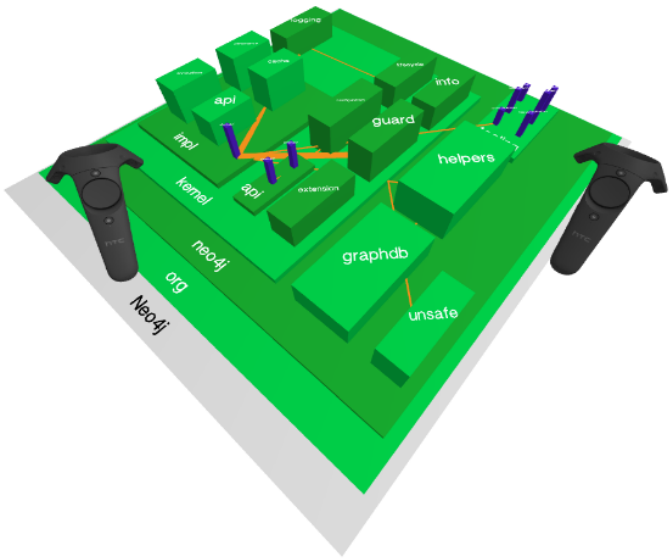


Figure 5: Screenshot of the VR extension of *ExplorViz* showing the application perspective and visualized VR controllers.

multi-user environment. As a proof of concept, they conducted a first usability evaluation with 22 subjects. The results of this evaluation revealed a good usability and thus constituted a valuable extension to *ExplorViz*. Recently, we performed a second evaluation focusing on the applicability of the approach for system and program comprehension tasks in teams. With 24 subjects, grouped into physically separated teams of two persons, they solved comprehension tasks collaboratively. First results indicated an efficient usage of the approach, which could offer an alternative to traditional 2D displays and interaction devices.

C. Architecture Conformance Checking

Software landscapes evolve over the time, and consequently, architecture erosion occurs. This erosion causes high maintenance and operation costs, thus performing architecture conformance checking (ACC) is an important task. ACC allows faster functionality changes and eases the adaptation to new challenges or requirements. Additionally, software architects can use ACC to verify a developed version against a previous modeled version. This can be used to check whether the current architecture complies with the specified architecture and allows to reveal constraint violations. An example architecture conformance visualization of a monitored software landscape against a modeled one is shown in Figure 6. The visualization illustrates missing or modified (colored in red), and additional (colored in blue) nodes and applications and related communication in-between for a software landscape. In this extension, a student developed an approach to perform ACC between a modeled software landscape consisting of applications using an editor and a monitored software landscape. This allows us to perform a visual comparison between both versions on an architectural level. In order to evaluate the extension, the student conducted a usability study with five participants, applying the model editor for a desired software landscape and performing ACC of a modeled software landscape against a monitored one. The results indicated a good user experience of the approach, although the usability of the editor could be improved.

D. Visualizing Architecture Comparison

Identifying architectural changes between two visualizations of a complex software application is a challenging task, which can be supported by appropriate tooling. Although *ExplorViz* visualizes the behavior and thus the runtime architecture of a software system, it is not possible to compare two versions. In this extension one student developed an approach to perform a visual software architecture comparison of two monitored applications, e.g., indicating removed or changed components or classes. This facilitates a developer to see at a glance which parts of the architecture have been added, deleted, modified, or remained unchanged between the two versions. Finally, an evaluation based on a qualitative usability study with an industrial partner was conducted. Five professional software engineers participated in the study and solved comparison tasks based on two different versions of their own developed software. The evaluation showed that the extension is applicable for solving architecture comprehension tasks with different versions within *ExplorViz*.

VII. Second Iteration: Restructured Architecture and New Process

As the evaluations at the end of the first iteration revealed some drawbacks, we decided to perform a second iteration of our modularization and modernization approach. After evaluating the first iteration we identified, among others, four major drawbacks, which are presented in the following.

- Extensibility & Integrability: Higher services had to perform several HTTP requests to obtain necessary information from variety of services.
- Code Quality & Comprehensibility: The coding quality was on a low level due to the lack of employed QA tools and rules.
- Software Configuration & Delivery: We needed to provide compiled Java files of all available backend extensions.
- Software Architecture Erosion & Accessibility: The configuration of the monitoring was still too difficult.

Our modularization approach started by dividing the old monolith into separated frontend and backend projects [26]. Since then, we further decomposed our backend into several microservices to address the problems stated in Section II. The resulting, restructured architecture is illustrated in Figure 7 and the new collaborative development process is described below. As reported in Section VI, the new architecture already improved the collaboration with new developers who realized new features as modular extensions.

A. Extensibility & Integrability

Frontend extensions are based on *Ember.js*'s addon mechanism. This approach works quite well for us as shown in Section VI. The backend, however, used the package scanning feature of *Jersey* to include extensions. The result of this

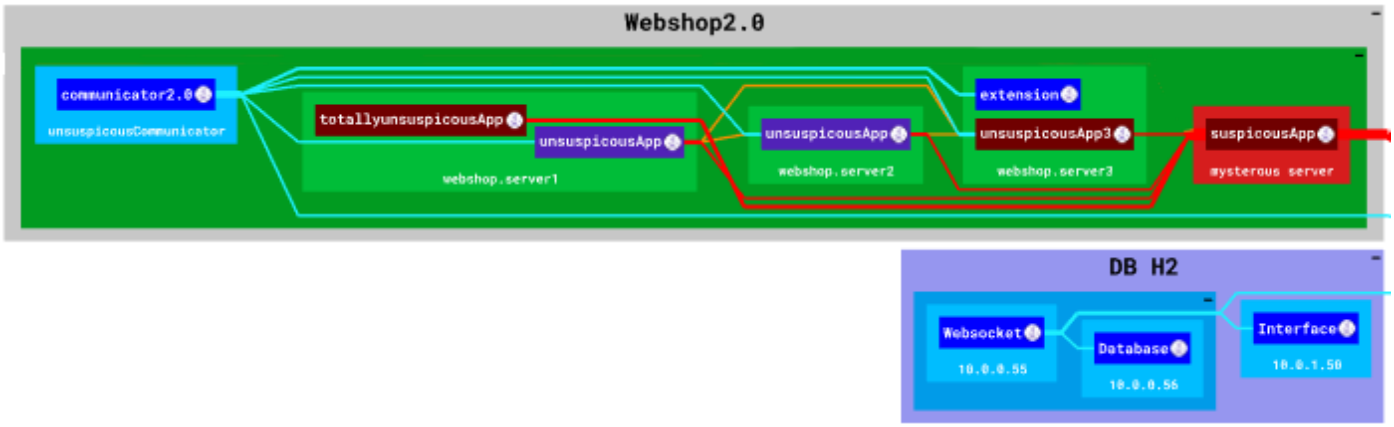


Figure 6: Screenshot of the architecture conformance checking extension of *ExplorViz*.

procedure was again an unhandy configuration of a monolithic application with high coupling of its modules. Therefore, we once again restructured the approach for our backend plug-in extensions. The extensions are now decoupled and represent separated microservices. As a result, each extension is responsible for its own data persistence and error handling. Due to the decomposition of the backend, we are left with multiple Uniform Resource Identifiers (URI). Furthermore, new extensions will introduce additional endpoints, therefore more URIs again. To simplify the data exchange handling based on those endpoints, we employ a common approach for microservice-based backends. The frontend communicates with an API gateway instead of several single servers, thus only a single base Uniform Resource Locator (URL) with well-defined, multiple URIs. This gateway, a *NGINX* reverse proxy [45], passes requests based on their URI to the respective proxied microservices, e.g., the *landscape-service*. Furthermore, the gateway acts as a single interface for extensions and offers additional features like caching and load balancing. Extension developers, who require a backend component, extend the gateway's configuration file, such that their frontend extension can access their complement. Some extensions must read data from different services. In the past, we used HTTP requests to periodically obtain this data. Each request was processed by the providing service, therefore introducing unnecessary load. The inter-service communication is now realized with the help of *Apache Kafka* [46]. *Kafka* is a distributed streaming platform with fault-tolerance for loosely coupled systems. We use *Kafka* for events that might be interesting for upcoming microservices. For example, the *landscape-service* consumes traces from the respective *Kafka* topic and produces a new landscape every tenth second for another topic. Microservices can consume the topic, obtain, and process the data in their custom way. As a result, the producing service does not have to process unnecessary HTTP requests, but simply fires its data and forgets it. Simple Create Read Update Delete (CRUD) operations on resources, e.g., users and their management, are provided by means of RESTful APIs by the respective microservices. The decomposition into several independent microservices and the new inter-service communication approach both facilitate low coupling in our system.

B. Code Quality & Comprehensibility

The improvements for code quality and accessibility, which were introduced in the first iteration of our modularization approach, showed a perceptible impact on contributor's work. For example, recurring students approved the easier access to *ExplorViz* and especially the obligatory exchange format *json:api*. However, we still lacked a common code style in terms of conventions and best practices. To achieve this and therefore facilitate maintainability, we defined compulsory rule sets for the quality assurance tools *Checkstyle* and *PMD*. In addition with *SpotBugs* [47], we impose their usage on contributors for Java code. For JS, we employ *ESLint* [48], i.e., a static analysis linter, with an *Ember.js* community-driven rule set. The latter contains best practices for *Ember.js* applications and rules to prevent programming flaws. In the future, we are going to enhance this rule set with our custom guidelines. Another aspect are CI tools. CI systems and tools are used to automate the compilation, building, and testing of software (systems). Software projects that employ CI, release twice as often, accept pull requests faster, and have developers who are less worried about breaking the build, compared to projects that do not use CI [49]. Therefore, employing CI tools is a good method to improve our development process even more. Consequently, we integrated the previously mentioned tools into our continuous integration pipeline configured in *TravisCI* [50]. More precisely, we employ *TravisCI* for *ExplorViz*'s core and any extension to build, test, and examine the code. Integrating the quality assurance tools allows us to define thresholds within the pipeline. If a threshold regarding quality assurance problems is exceeded, the respective *TravisCI* build will fail and the contributor is notified by mail. A similar build is started for each pull request that we receive on *Github* for the now protected master branch. Therefore, contributors are forced to create a new branch or fork *ExplorViz* to implement their enhancement or bug fix and eventually submit a pull request.

C. Software Configuration & Delivery

One major problem of *ExplorViz Legacy* was the necessary provision of software configurations for different use cases. The first iteration of modularization did not entirely solve this problem. The backend introduced a first approach for an

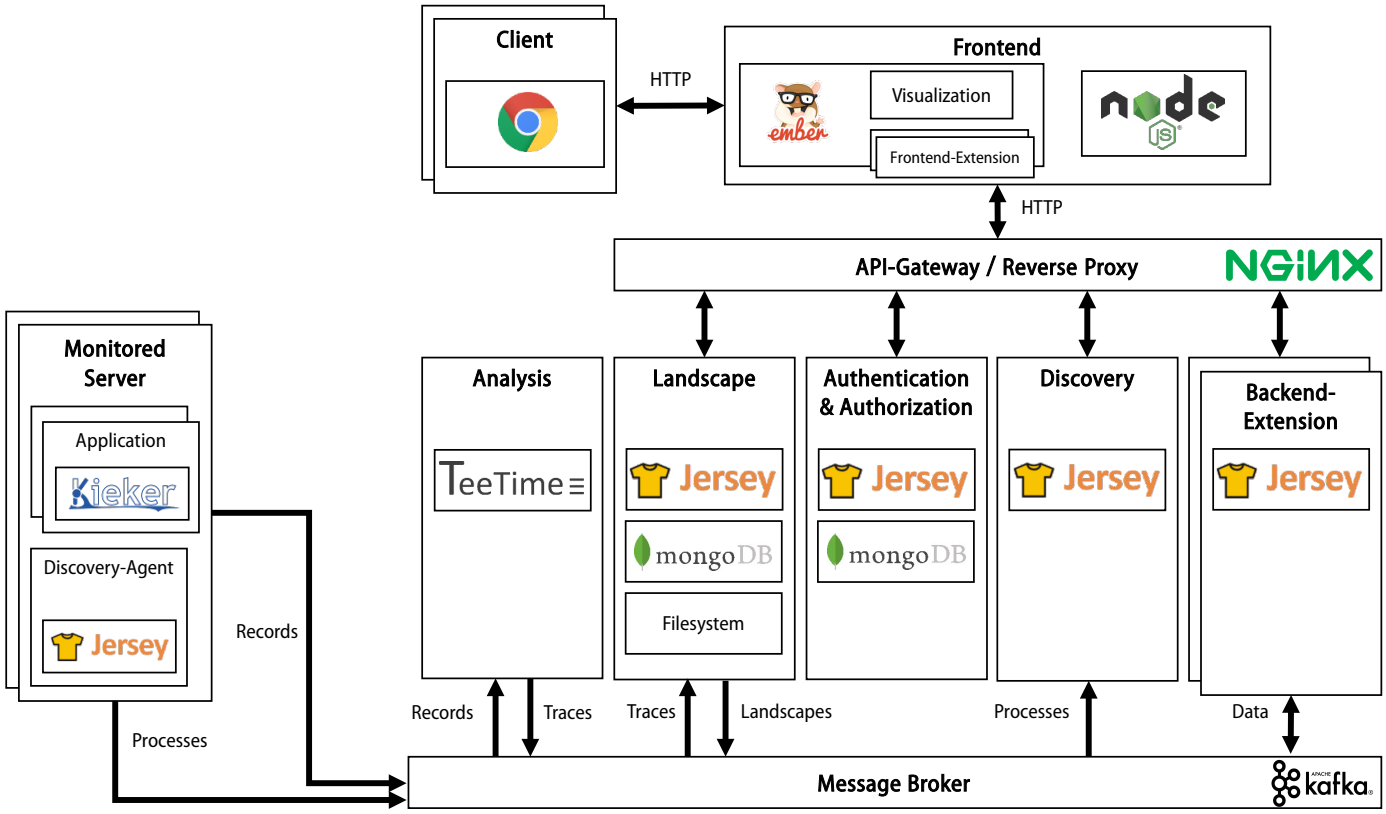


Figure 7: Architectural overview and software stack of *ExplorViz* (after the second iteration).

integration of extensions, but their delivery was cumbersome. Due to the tight coupling at source code level we had to provide the compiled Java files of all extensions for download. Users had to copy these files to a specific folder in their already deployed *ExplorViz* backend. Therefore, configuration alterations were troublesome. With the architecture depicted in Figure 7 we can now provide a jar file for each service with an embedded web server. This modern approach for Java web applications facilitates the delivery and configuration of *ExplorViz*'s backend components. In the future, we are going to ship ready-to-use *Docker* images for each part of our software. The build of these images will be integrated into our CI pipeline. Users are then able to employ docker-compose files to achieve their custom *ExplorViz* configuration or use a provided docker-compose file that fits their needs. As a result, we can provide an alternative, easy to use, and exchangeable configuration approach that essentially only requires a single command line instruction. The frontend requires another approach, since (to the best of our knowledge) it is not possible to install an *Ember.js* addon inside of a deployed *Ember.js* application. We are currently developing a build service for users that ships ready-to-use, pre-built configurations of our frontend. Users can then download and deploy these pre-built packages. Alternatively, these configurations will also be usable as *Docker* containers.

D. Software Architecture Erosion & Accessibility

One of our initial problems was the partitioning of our code base and the resulting software architecture erosion. We think that both employed frameworks, *Ember.js* and *Jersey*, matter when it comes to this problem. *Ember.js* is well documented and there are many examples on how to solve a problem with the framework. Due to its JS nature, we can easily introduce and use modern features in web development. Furthermore, *Ember.js* introduces recognizable and reusable structures which facilitate the development. For the *Jersey* backend, we again provide a sample project that contributors can use for a start. The project is runnable and shows how to use *Kafka* and the HTTP client for different needs. *ExplorViz* uses the monitoring framework *Kieker* to obtain monitoring data. These so called Records are then processed by the analysis component of our software. The setup of *Kieker* is extensive, but also quite complex for untrained users. Since we are dealing with many students, we were in need of a solution to circumvent this drawback. We developed an external component with a frontend and backend extension that simplifies the monitoring setup for users. The so called discovery agent searches for running Java processes in the encompassing operating system and sends its data to the related discovery backend extension. The frontend discovery extension visualizes the gathered data and provides Graphical User Interface (GUI) forms for users to start and stop the monitoring of found processes. Ultimately, the resulting discovery mode was successful in internal tests and we integrated it as a core feature.

VIII. Third Iteration: Achieving an entire Microservice Architecture

The second iteration of our modernization process introduced multiple microservices for different backend logic. For example, each backend extension was build as a separate source code project and deployed as Java jar file. This introduced advantages, among others, for the configuration of *ExplorViz* as described in Section VII. Since then, we further refined our microservice decomposition. The current architecture, after performing a third iteration of our modularization approach, is illustrated in Figure 8. Additionally, we revised the ubiquitous problems revealed within the evaluation, as we did in the previous iterations. Thus, we identified, among others, three major drawbacks, which are presented in the following.

- Extensibility & Integrability: Implementing extensions against specific backend or frontend versions was difficult.
- Code Quality & Comprehensibility: Collaborators like students, did not have enough documentation to efficiently contribute to our project.
- Software Configuration & Delivery: The testing and release management was still cumbersome due to a vast number of artifacts.

A. Extensibility & Integrability

Both previous iterations shared the problem that collaborators had to implement their feature or extension against the latest version of *ExplorViz*. To circumvent this drawback, we now push the backend build artifacts of the *TravisCI* build pipeline as snapshots to *Sonatype* [51], i.e., an online maven repository for unsigned artifacts. Furthermore, we use *Github* releases to version *ExplorViz*. These releases follow a documented release management process. As a result, release descriptions and names share a common theme. In general, *Github* releases use Git tags to reference the specific Git commit that represents the release. We use these resulting Git tags for versioning. The tags are picked up by our CI pipeline and are used to name the *Sonatype* snapshots. As a result, contributors can now select specific (intermediate) versions to implement against.

After employing the second iteration of our modernization for some time with different configurations, we observed performance issues regarding the *landscape-service*. This service continuously built our hierarchical landscape model, provided the latest snapshot of the model via a HTTP API, and returned previous snapshots upon incoming HTTP requests. We identified that we could decompose these functionalities into separated microservices to distribute the load on one hand and gain a better performance on the other hand. The decoupling of the *landscape-service* can be seen in Figure 8. Frontend extensions now register at the *broadcast-service* to receive server-sent events (SSE), which contain the latest landscape model snapshot. Furthermore, specific snapshots can be requested at the *history-service*. This microservice is responsible for storing landscape model snapshots.

B. Code Quality & Comprehensibility

Introducing static analysis tools to our CI pipeline showed improvements of *ExplorViz*'s code style. The automatic CI build for *Github* pull requests highlights flaws and allows us to impose refactoring before merging the code. This is also used for collaborators' extensions. Now, the remaining part to improve the overall code quality was testing the source code and the integration of components. We observed that collaborators had less problems with testing frontend extensions than with testing the related backend project. We think that is due to the *Ember.js* documentation and the huge number of already existing open source projects, which already show how one can comprehensively test *Ember.js* projects. Therefore, we wrote sample unit, integration, and API tests for our microservices, which students can use as foundation to test their own written code. By choosing these three categories of tests, we now cover testing at source code and API level. All these tests are automatically executed as part of our CI pipeline. Furthermore, when a tests requires other running services, e.g., the reverse proxy, these services are (if necessary) build and executed by means of a *Docker* container.

To ease the development for collaborators, we wrote supplemental guides on best practices, design ideas, and specifications. These can be found in our public *Github* documentation wiki [52]. Furthermore, our CI pipeline now automatically builds the latest API documentation (*JavaDoc* for the backend and *YUIDoc* for the frontend). The resulting websites are deployed by means of *Github* pages, i.e., public websites based on the content of Git repositories. We additionally employ *Swagger* [53], an interactive API development editor and UI, to document our HTTP APIs. The tool is automatically started when a microservice is started in development mode.

C. Software Configuration & Delivery

ExplorViz enables users and developers to use extensions on demand by providing the build artifacts for every (release) version. We now facilitate *ExplorViz*' configuration with the help of *Docker* images. After pushing the build artifacts to *Sonatype* in the CI pipeline, we subsequently build a *Docker* image for each service and push it to *Docker Hub*. Therefore, users and collaborators can use the publicly hosted *Docker* images to easily create their custom deployment environment with *Docker*.

We build upon this process and now provide ready-to-use docker-compose files for release versions of *ExplorViz*. These configurations allow users to start the core features of *ExplorViz* with only a single command. This approach is also used in the development phase. Since *ExplorViz* requires auxiliary software, i.e., database management systems, *Apache Kafka*, and the reverse proxy *NGINX*, we now provide a docker-compose file to start the mandatory, already configured software stack for development. As a result, collaborators do not need to read different instructions on how to start specific software, but only need to start a set of *Docker* containers with the help of the docker-compose file.

Figure 8 shows that we replaced our employed reverse proxy *NGINX* with *Traefik* [54]. The reverse proxy *NGINX* uses a static configuration file to define its routing. As a result, *ExplorViz* users needed to update this configuration or use a

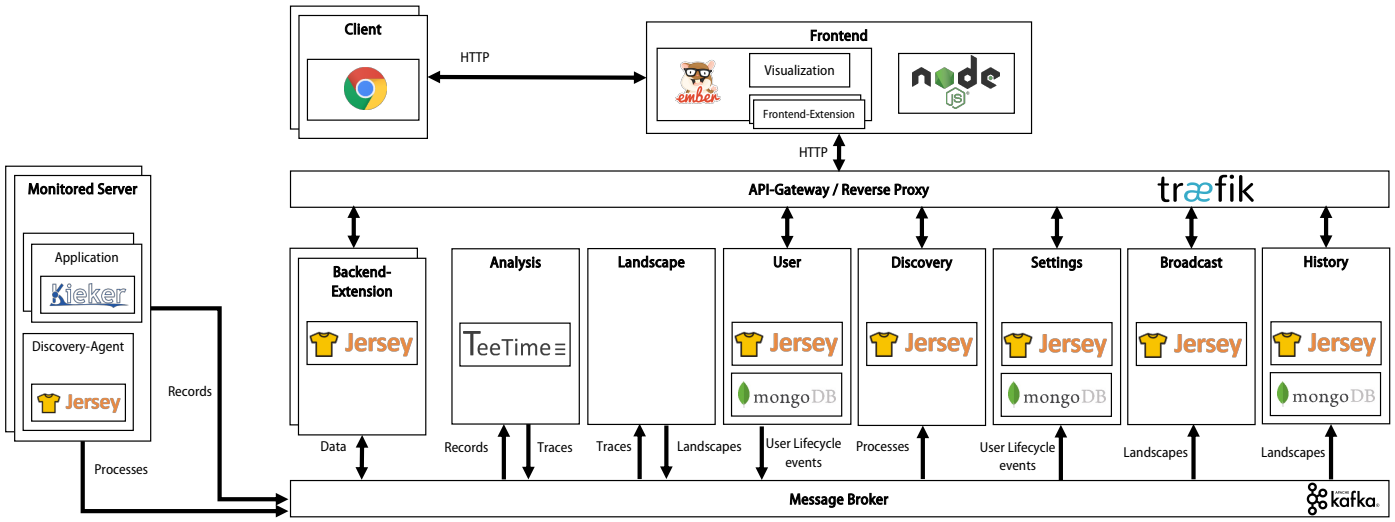


Figure 8: Current architectural overview and software stack of *ExplorViz* (after the third iteration).

provided version to enable an installed or developed extension. This was quite cumbersome and potentially deterred users to try out extensions. With *Traefik* we can now use labels, i.e., metadata for *Docker* objects, to define the routing at docker-compose level. Therefore, the routing of the reverse proxy can be easily extended or changed.

IX. Related Work

In the area of software engineering, there are many papers that perform a software modernization in other contexts. Thus, we restrict our related work to approaches, which focus on the modernization of monolithic applications towards a microservice architecture. Compared to frequently performed software modernizations, we did not reconstruct the underlying software architecture, since it was not our goal to keep the obsolete monolithic architecture provided by *GWT*. Furthermore, we did not need to apply multiple refactoring iterations to modernize our software system. Instead, we successfully performed three iterations of our modularization and modernization process *CORAL* in order to continuously improve our software architecture and collaborative development process.

Villamizar et al. [55] evaluate monolithic and microservice architectures regarding the development and cloud deployment of enterprise applications. Their approach addresses similar elements to our modernization process. They employed modern technologies for separating microservices, e.g., Java in the backend and JS in the frontend, like we did. Contrary to their results, we did not face any of the mentioned problems during the migration, like failures or timeouts. In [56] an approach regarding the challenges of the modernization of legacy J2EE applications was presented. They employ static code analysis to reconstruct architectural diagrams, which then can be used as a starting point during a modernization process. In contrast to our approach there was no need to reconstruct the software architecture, because we wanted to modernize it from the beginning due to previously mentioned drawbacks. Thus, we split our application based on our knowledge into several microservices and developed a communication concept based on a message broker. Carrasco et. al [34] present a survey of architectural smells during the modernization towards a microservice architecture. They identified nine common pitfalls in terms of bad smells and provided potential solutions for them. *ExplorViz Legacy* was also covered by this survey and categorized by the "Single DevOps toolchain" pitfall. This pitfall concerns the usage of a single toolchain for all microservices. Fortunately, we addressed this pitfall since their observation during their survey by employing independent toolchains by means of pipelines within our continuous integration system for the backend and frontend microservices.

Knoche and Hasselbring [31] present a migration process to decompose an existing software system into several microservices. Additionally, they report from their gained experiences towards applying their presented approach in a legacy modernization project. Although their modernization drivers and goals are similar to our procedure, their approach features a more abstract point of view on the modernization process. Furthermore, they focus on programming language modernization and transaction systems. In [4], the authors present an industrial case study concerning the evolution of a long-living software system, namely a large e-commerce application. The addressed monolithic legacy software system was replaced by a microservice-based system. Compared to our approach, this system was completely rebuilt without retaining code from the (commercial) legacy software system. Our focus is to facilitate the collaborative development of open source software and also addresses the development process. We successfully developed our pipeline towards CI for all microservices mentioned in Section VII to minimize the release cycles and offer development snapshots.

A different approach to perform a modernization of a monolithic application is presented in [57]. They employed a Domain-Driven Design (DDD) based approach to decomposition their software system into services. Afterwards, they integrated the services with an Enterprise Bus and orchestrated the services on the basis of Docker Compose and Swarm. In contrast to their approach, we did not perform a decomposition of our monolithic application based on DDD. Instead, we performed a decomposition based on backend and frontend

logic within our first iteration and refined it later. Additionally, we employ Docker images for the deployment of *ExplorViz* and do not use Docker swarm. Chen et. al [58] present a top-down based dataflow-driven approach as an alternative decomposition method. More precisely, they developed a dataflow-driven decomposition algorithm, which operates on the basis of a constructed dataflow diagram modeling the business logic of the software system. In the next step, the dataflow diagram is compacted based on similar operations with the same type of output data. Finally, microservice candidates are identified and extracted. In comparison, our approach does not facilitate the usage of an algorithm which aids the decomposition process and identifies microservice candidates. In detail, we propose a collaborative-oriented, iterative process, which contains multiple steps and also addresses the involved development process.

X. Conclusion

In the following, we conclude our paper and present a summary, depict lessons learned, and give an outlook for future work.

A. Summary

In this paper, we reported on our modularization and modernization process of the open source research software *ExplorViz*, moving from a monolithic architecture towards a microservice architecture with the primary goal to ease the collaborative development, especially with students. We described technical and development process related drawbacks of our initial project state until 2016 in *ExplorViz Legacy* and illustrated our modularization process and architecture. The process included not only a decomposition of our web-based application into several components, but also a technical modernization of applied frameworks and libraries. Driven by the goal to easily extend our project in the future and facilitate a contribution by inexperienced collaborators, we offer a plug-in extension mechanism for our core project, both for backend and frontend. On the basis of *ExplorViz Legacy*, we employed our iterative, collaborative modularization and reengineering process *CORAL* as a guidance through our modularization and performed three successful iterations to *ExplorViz Legacy* until we reached a sufficient state.

After our first iteration, we realized our modularization process and architecture in terms of a proof-of-concept implementation and evaluated it afterwards by the development of several extensions of *ExplorViz*. Each of these extensions was developed by students and evaluated afterwards, in each case by at least a usability study. The results showed an overall good usability of each extension. In the case of our developed application discovery extension, we integrated it into our core project based on the high-quality of the extension in addition to the good usability and time saving aspect when instrumenting applications with *Kieker*. As the results of the the modularization process were not sufficient yet, we performed a second iteration featuring a first microservice architecture. More precisely, the iteration led to several independent deployable services bundled with inter-service communication handled via the message broker *Kafka* and requests from the frontend towards the backend are passed through our reverse-proxy in form of *NGINX*. Furthermore, we enhanced our development and build process towards a more collaborative manner. Unfortunately, we were not satisfied with the results of the second iteration, because some services were still very large and poorly maintainable. Thus, we needed to perform a further decoupling of them. Additionally, we recognized that our release management and CI processes, as well as our documentation, still needed to be improved. Consequently, with these drawbacks in mind, we performed a third iteration, after which we achieved a fully decoupled microservice architecture, consisting of a set of self-contained systems and well-defined interfaces in-between. The inter-service communication is still handled through *Kafka*. Additionally, we replaced our reverse-proxy with *Traefik* for handling requests from the frontend towards the backend. For the release management and documentation, we further optimized our CI pipeline regarding *Docker* images and supplemental (API) documentation for both developers and users.

B. Lessons Learned

The lessons learned while applying our *CORAL* approach to *ExplorViz* within three successfully performed iterations are summarized in the following. Performing software development for research is a challenging task, especially when a large number of inexperienced students is involved. In our experience, providing an extensible software architecture is a crucial task for open source (research) projects. Furthermore, extension points, i.e., interfaces, should be well-documented to ease the development of extensions. Additionally, the overall software development process should base on accessible documentation for all stages during the development for all collaborators. This includes documentation of the employed software architecture and the extension mechanisms, but should also cover best practices, hints, or lessons learned.

Regarding software quality, especially with respect to maintainability, we recommend the usage of software quality tools. Static analysis tools such as *Checkstyle* or *PMD* in combination with configured common code styles support developers directly while they code. This way, common programming flaws can be avoided in the committed source code and thus result in less bugs and required bug fixes. Also, we suggest the setup and usage of CI pipelines that allow a project to automate their testing, code quality checking, and software building. Thereby, the complete build cycle can be tested periodically. If the building is triggered by commits, developers also get an early feedback if something went wrong.

Providing *Docker* images of *ExplorViz* provides great value. Developers and users are able to use pre-configured images of our software for specific use cases, which may be based on different versions. This approach also eases the release management process and facilitates developers to test and adapt their extension to upcoming versions.

C. Future Work

In the future, we are planning to evaluate our finalized project, especially in terms of developer collaboration. Additionally, we plan to move from our CI pipeline towards a continuous delivery (CD) environment. Thus, we expect to further decrease the interval between two releases and allow users to try out new versions, even development snapshots, as soon as possible. Furthermore, we plan to use architecture

recovery tools like [59] for refactoring or documentation purposes in upcoming versions of *ExplorViz*. Recently, we applied *ExplorViz* within case study, where we successfully performed a microservice decomposition with static and dynamic analysis of a monolithic application [60]. As a result, we plan to investigate, if we could enhance our *CORAL* approach with the applied decomposition process for future projects.

References

[1] C. Zirkelbach, A. Krause, and W. Hasselbring, "Modularization of Research Software for Collaborative Open Source Development," in *Proceedings of the The Ninth International Conference on Advanced Collaborative Networks, Systems and Applications (COLLA 2019)*, Jun. 2019, pp. 1–7.

[2] C. Goble, "Better Software, Better Research," *IEEE Internet Computing*, vol. 18, no. 5, pp. 4–8, Sep. 2014.

[3] A. Johanson and W. Hasselbring, "Software engineering for computational science: Past, present, future," *Computing in Science & Engineering*, vol. 20, no. 2, pp. 90–109, Mar. 2018. DOI: 10.1109/MCSE.2018.021651343.

[4] W. Hasselbring and G. Steinacker, "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce," in *Proceedings of the IEEE International Conference on Software Architecture Workshops (ICSAW)*, Apr. 2017, pp. 243–246. DOI: 10.1109/ICSAW.2017.11.

[5] P. D. Francesco, P. Lago, and I. Malavolta, "Migrating Towards Microservice Architectures: An Industrial Survey," in *Proceedings of the IEEE International Conference on Software Architecture (ICSA)*, Apr. 2018, pp. 29–38.

[6] F. Fittkau, A. Krause, and W. Hasselbring, "Software landscape and application visualization for system comprehension with ExplorViz," *Information and Software Technology*, vol. 87, pp. 259–277, Jul. 2017. DOI: doi:10.1016/j.infsof.2016.07.004.

[7] F. Fittkau, S. Roth, and W. Hasselbring, "ExplorViz: Visual runtime behavior analysis of enterprise application landscapes," in *Proceedings of the 23rd European Conference on Information Systems (ECIS 2015 Completed Research Papers)*, AIS Electronic Library, May 2015, pp. 1–13. DOI: 10.18151/7217313.

[8] R. Heinrich, C. Zirkelbach, and R. Jung, "Architectural Runtime Modeling and Visualization for Quality-Aware DevOps in Cloud Applications," in *Proceedings of the IEEE International Conference on Software Architecture Workshops (ICSAW)*, Apr. 2017, pp. 199–201.

[9] R. Heinrich, R. Jung, C. Zirkelbach, W. Hasselbring, and R. Reussner, "An Architectural Model-Based Approach to Quality-aware DevOps in Cloud Applications," in *Software Architecture for Big Data and the Cloud*, I. Mistrik, R. Bahsoon, N. Ali, M. Heisel, and B. Maxim, Eds., Cambridge: Elsevier, Jun. 2017, pp. 69–89.

[10] F. Fittkau, A. Krause, and W. Hasselbring, "Hierarchical software landscape visualization for system comprehension: A controlled experiment," in *Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*, IEEE, Sep. 2015, pp. 36–45. DOI: 10.1109/VISSOFT.2015.7332413.

[11] F. Fittkau, S. Finke, W. Hasselbring, and J. Waller, "Comparing Trace Visualizations for Program Comprehension through Controlled Experiments," in *Proceedings of the 23rd IEEE International Conference on Program Comprehension (ICPC 2015)*, May 2015, pp. 266–276. DOI: 10.1109/ICPC.2015.37.

[12] Open Source Software Community, *Google Web Toolkit Project (GWT)*, version 2.8.2, last accessed: 2020.05.31. [Online]. Available: http://www.gwtproject.org.

[13] H. Al-Kilidar, K. Cox, and B. Kitchenham, "The use and usefulness of the iso/iec 9126 quality standard," in *Proceedings of the International Symposium on Empirical Software Engineering, 2005.*, 2005, pp. 126–132.

[14] Open Source Software Community, *Checkstyle*, version 8.10, last accessed: 2020.05.31. [Online]. Available: http://checkstyle.sourceforge.net.

[15] ——, *PMD*, version 6.10.0, last accessed: 2020.05.31. [Online]. Available: https://pmd.github.io.

[16] Eclipse Foundation, *AspectJ*, version 1.8.5, last accessed: 2020.05.31. [Online]. Available: https://www.eclipse.org/aspectj.

[17] Open Source Software Community, *H2*, version 1.4.177, last accessed: 2020.05.31. [Online]. Available: http://www.h2database.com.

[18] B. Upadhyaya, Y. Zou, H. Xiao, J. Ng, and A. Lau, "Migration of SOAP-based services to RESTful services," in *Proceedings of the 13th IEEE International Symposium on Web Systems Evolution (WSE)*, Sep. 2011, pp. 105–114.

[19] S. Vinoski, "RESTful Web Services Development Checklist," *IEEE Internet Computing*, vol. 12, no. 6, pp. 96–95, Nov. 2008, ISSN: 1089-7801.

[20] R. Kazman, M. Klein, and P. Clements, "Atam: Method for architecture evaluation," Carnegie-Mellon Software Engineering Institute, University Pittsburgh, PA, Tech. Rep., 2000.

[21] H. Koziolek, "Sustainability evaluation of software architectures: A systematic review," in *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*, ser. QoSA-ISARCS '11, Boulder, Colorado, USA: ACM, 2011, pp. 3–12, ISBN: 978-1-4503-0724-6. DOI: 10.1145/2000259.2000263.

[22] J. Knodel and M. Naab, "Software architecture evaluation in practice: Retrospective on more than 50 architecture evaluations in industry," in *2014 IEEE/IFIP Conference on Software Architecture*, Apr. 2014, pp. 115–124. DOI: 10.1109/WICSA.2014.37.

[23] J. Knodel and M. Naab, *Pragmatic Evaluation of Software Architectures*. Springer, 2016.

[24] M. Dick and S. Naumann, "Enhancing software engineering processes towards sustainable software product design," in *Integration of Environmental Information in Europe*, K. Greve and A. B. Cremers, Eds., Aachen: Shaker Verlag, 2010.

[25] P. Clarke and R. V. O'Connor, "An approach to evaluating software process adaptation," in *Software Process Improvement and Capability Determination*, R. V. O'Connor, T. Rout, F. McCaffery, and A. Dorling, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 28–41, ISBN: 978-3-642-21233-8.

[26] C. Zirkelbach, A. Krause, and W. Hasselbring, "On the Modernization of ExplorViz towards a Microservice Architecture," in *Combined Proceedings of the Workshops of the German Software Engineering Conference 2018*, vol. Online Proceedings for Scientific Conferences and Workshops, Ulm, Germany: CEUR Workshop Proceedings, Feb. 2018.

[27] A. Tang, M. Razavian, B. Paech, and T. Hesse, "Human Aspects in Software Architecture Decision Making: A Literature Review," in *Proceedings of the IEEE International Conference on Software Architecture (ICSA)*, Apr. 2017, pp. 107–116.

[28] D. M. Le, P. Behnamghader, J. Garcia, D. Link, A. Shahbazian, and N. Medvidovic, "An Empirical Study of Architectural Change in Open-Source Software Systems," in *Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR)*, May 2015, pp. 235–245.

[29] D. M. Le, D. Link, A. Shahbazian, and N. Medvidovic, "An Empirical Study of Architectural Decay in Open-Source Software," in *Proceedings of the IEEE International Conference on Software Architecture (ICSA)*, Apr. 2018, pp. 176–17 609.

[30] H. Knoche and W. Hasselbring, "Drivers and barriers for microservice adoption – a survey among professionals in Germany," *Enterprise Modelling and Information Systems Architectures (EMISAJ) – International Journal of Conceptual Modeling*, vol. 14, no. 1, pp. 1–35, 2019. DOI: 10.18417/emisa.14.1.

[31] H. Knoche and W. Hasselbring, "Using Microservices for Legacy Software Modernization," *IEEE Software*, vol. 35, no. 3, pp. 44–49, May 2018, ISSN: 0740-7459.

[32] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, Today, and Tomorrow," in *Present and Ulterior Software Engineering*. Springer International Publishing, 2017, pp. 195–216.

[33] N. Alshuqayran, N. Ali, and R. Evans, "A Systematic Mapping Study in Microservice Architecture," in *Proceedings of the 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, Nov. 2016, pp. 44–51.

[34] A. Carrasco, B. v. Bladel, and S. Demeyer, "Migrating Towards Microservices: Migration and Architecture Smells," in *Proceedings of the 2nd International Workshop on Refactoring*, ser. IWoR 2018, Montpellier, France: ACM, 2018, pp. 1–6.

[35] Oracle, *Jersey Project*, version 2.27, last accessed: 2020.05.31. [Online]. Available: https://jersey.github.io.

[36] A. van Hoorn, J. Waller, and W. Hasselbring, "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis," in *Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*, ACM, Apr. 2012, pp. 247–248.

[37] Ember Core Team, *Ember.js*, version 3.6.0, last accessed: 2020.05.31. [Online]. Available: https://www.emberjs.com.

[38] Joyent, *Node.js*, version 10.15.0, last accessed: 2020.05.31. [Online]. Available: https://nodejs.org.

[39] Open Source Software Community, *json:api*, version 1.0.0, last accessed: 2020.05.31. [Online]. Available: https://jsonapi.org.

[40] A. Krause, C. Zirkelbach, and W. Hasselbring, "Simplifying Software System Monitoring through Application Discovery with ExplorViz," in *Proceedings of the Symposium on Software Performance 2018: Joint Developer and Community Meeting of Descartes/Kieker/Palladio*, Nov. 2018.

[41] L. Garber, "Gestural Technology: Moving Interfaces in a New Direction," *Computer*, vol. 46, no. 10, pp. 22–25, 2013, ISSN: 0018-9162. DOI: 10.1109/MC.2013.352. [Online]. Available: http://dx.doi.org/10.1109/MC.2013.352.

[42] F. Fittkau, A. Krause, and W. Hasselbring, "Exploring Software Cities in Virtual Reality," in *Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*, 2015, pp. 130–134. DOI: 10.1109/VISSOFT.2015.7332423. [Online]. Available: http://dx.doi.org/10.1109/VISSOFT.2015.7332423.

[43] A. Elliott, B. Peiris, and C. Parnin, "Virtual Reality in Software Engineering: Affordances, Applications, and Challenges," in *Proceedings of the 37th IEEE International Conference on Software Engineering*, vol. 2, May 2015, pp. 547–550. DOI: 10.1109/ICSE.2015.191.

[44] C. Zirkelbach, A. Krause, and W. Hasselbring, "Hands-On: Experiencing Software Architecture in Virtual Reality," Kiel University, Research Report, Jan. 2019. [Online]. Available: http://oceanrep.geomar.de/45728/.

[45] NGINX, *NGINX*, version 1.15.8, last accessed: 2020.05.31. [Online]. Available: http://nginx.org.

[46] Apache Software Foundation, *Apache Kafka*, last accessed: 2020.05.31. [Online]. Available: https://kafka.apache.org.

[47] Open Source Software Community, *Spotbugs*, version 3.1.10, last accessed: 2020.05.31. [Online]. Available: https://spotbugs.github.io.

[48] ESLint Team, *ESLint*, version 5.12.0, last accessed: 2020.05.31. [Online]. Available: https://eslint.org.

[49] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Sep. 2016, pp. 426–437.

[50] Open Source Software Community, *TravisCI*, last accessed: 2020.05.31. [Online]. Available: https://travis-ci.org.

[51] ——, *Sonatype*, last accessed: 2020.05.31. [Online]. Available: https://oss.sonatype.org.

[52] ExplorViz Team, *ExplorViz Developer and User Documentation Wiki*, last accessed: 2020.05.31. [Online]. Available: https://github.com/ExplorViz/Docs/wiki.

[53] Open Source Software Community, *Swagger*, last accessed: 2020.05.31. [Online]. Available: https://swagger.io.

[54] ——, *Traefik*, last accessed: 2020.05.31. [Online]. Available: https://containo.us/traefik/.

[55] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *Proceedings*

of the 10th Computing Colombian Conference (10CCC), Sep. 2015, pp. 583–590.

[56] D. Escobar, D. Cárdenas, R. Amarillo, E. Castro, K. Garcés, C. Parra, and R. Casallas, "Towards the understanding and evolution of monolithic applications as microservices," in *Proceedings of the XLII Latin American Computing Conference (CLEI)*, Oct. 2016, pp. 1–11.

[57] J. Gouigoux and D. Tamzalit, "From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture," in *Proceedings of the IEEE International Conference on Software Architecture Workshops (ICSAW)*, Apr. 2017, pp. 62–65. DOI: 10.1109/ICSAW.2017.35.

[58] R. Chen, S. Li, and Z. Li, "From Monolith to Microservices: A Dataflow-Driven Approach," in *Proceedings of the 24th Asia-Pacific Software Engineering Conference (APSEC)*, 2017, pp. 466–475.

[59] G. Granchelli, M. Cardarelli, P. D. Francesco, I. Malavolta, L. Iovino, and A. D. Salle, "MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems," in *Proceedings of the IEEE International Conference on Software Architecture Workshops (ICSAW)*, Apr. 2017, pp. 298–302.

[60] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, and D. Kröger, "Microservice Decomposition via Static and Dynamic Analysis of the Monolith," in *Proceedings of the IEEE International Conference on Software Architecture (ICSA 2020)*, Mar. 2020. [Online]. Available: https://arxiv.org/abs/2003.02603.

An Integrated Syntax/Semantics Representational Framework for Natural Language Processing: Theory and Applications

Carlos Seror
Independent researcher
Valencia, Spain
email: serorcarlos@gmail.com

Abstract—**Language can be described as a set of encoding/decoding rules whereby a receiver is prompted to locally reconstruct a relevant part of a sender's representation, intended as an update of the receiver's representation. In this paper, a representational framework is proposed for such description, based on (a) the cognitive feature of spontaneous categorization, which leads to a formal description of data referencing as a disambiguation process, (b) the identification of a number of irreducible structures underlying perceptual categories, consistent with the notion of semantic primitives. The general algebra describing data referencing could be seen as a universal syntax from which conventional languages can be derived and, conversely, into which conventional languages can be parsed. On the other hand, the semantic structures identified will be formalized into a denotational algebra reminiscent of, but not identical to, the topology of open sets. Both formalisms will be shown to converge into a representational framework having the two-way capability to generate language and encode meaning. The framework thus proposed reflects a qualitative approach to language, and therefore radically differs from Shannon's quantitative approach to communication. As a demonstration of its potential, the last sections will sketch two practical applications, i.e., (i) a representational interpretation of databases, and (ii) a tool to enhance deafblind people's cognitive world.**

Keywords-data representation; categories; natural language; syntax; semantic structures; data referencing; databases; deafblind people

I. INTRODUCTION

A list of previous attempts to decode and/or generate human language would be huge. However, aside from their generally remarkable intellectual merit, none of those attempts seems to have fully succeeded to date [6][23]. Furthermore, a focus on the structural aspects of syntax tends to overlook the nature of language as an information tool, and more specifically as a conveyor of meaning [30]. Besides, there seems to be a remarkable polysemy between information as static data content (for example, [44] along the lines of concept theory) and Shannon's dynamic interpretation, based on the transmission and acquisition of data streams.

Shannon's classical paper [41] explicitly disregards meaning as irrelevant to a quantitative approach to communication. Shannon's information theory is about symbol strings, and views communication merely as a quantifiable streaming of symbols, i.e., a succession of elementary information events. It does not, however, explicitly formalize the notion of information event, which should arguably be considered a first essential step to comprehend the nature of language as a communication tool. In this paper, a general definition of an information event will be proposed, and shown to lead to a comprehensive description of language, both in terms of syntax and semantics. That description will be derived from a representational approach to data structures based on (a) the cognitive feature of spontaneous categorization, implicitly used by conventional languages' users, and (b) the identification of structures underlying perceptual categories, which can be formally described and generalized, thereby opening the way to an objective theory of semantics.

Section II will discuss Shannon's quantitative approach from the standpoint of meaning, and propose a representational definition of an information event. In Section III, the concept of data aggregate will be introduced, and endowed with a simple structure based on the logical connectives $\wedge$, $\vee$. A string syntax derived from the resulting expressions will be shown to be consistent with conventional languages. In Section IV, the scope of the connectives will be enlarged to derive more general structures. Based on such structures, the two key components of an information event will be formally described in Section V. As a practical application, a parser will be sketched in Section VI, followed in Section VII by a formal description of the spontaneous categorization feature. Sections VIII and IX will deal with the relations between representations and meaning, based on the spatial adjacency relation. The identification of denotational structures underlying perceptual categories will be the subject of Sections X and XI, illustrated in Section XII with three elementary semantic structures. Section XIII will propose an interpretation of databases as category clusters. Finally, Section XIV will assess the potential of the proposed framework to facilitate the communication and/or enhance the cognitive universe of deafblind users.

II. INFORMATION EVENTS

Parrots can speak, but cannot really talk. This difference is arguably the key to what we understand as language. Parrots are able to send and receive information as a sequence of vowels and consonants, which is strictly

sufficient to reconstruct a message, but hardly the information a human would expect from an act of communication. Therefore, a key question is: what do we precisely mean when we say 'information'?

Surprisingly, there seems to be no generally agreed definition of information [31]. In 1948, Shannon's classic paper addressed information as the "signature" of communication, but Shannon was an engineer and his basic concern was to ensure that two parties located at either end of a telephone line could convey their messages with an acceptable degree of noise. Even if one of the parties was a parrot.

At the beginning of his paper, Shannon stated: "Frequently the messages have meaning; that is, they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem [i.e., reproducing at one point more or less accurately a message selected at another point]" [41]. To Shannon, information reaches its destination when the message unequivocally does. The messenger's mission is to deliver messages, not understand them. However, if my parrot and I are in the living room and in the kitchen somebody shouts "Fire!", it can hardly be argued that both the parrot and I have received the same information. Therefore, the key issue to be addressed in human communication is: how do we convey meaning?

A formal definition of the communication process should be a key starting point to answer that question. A communication event could be described as a process whereby a sender encodes a number of instructions intended to replicate some part of its data repository. Communication could be said to succeed when the replication succeeds and results in an enlargement of the receiver's data repository.

We may think of a communication process as an assembling process: first, we point to some location in a data aggregate, then we assign some new item to that location. The item to be assembled is the information item the sender wants to convey, based on the conjecture that the item is missing in the receiver's repository. Such assembling operation can be formally expressed as a definition of an information event. For a communication event to take place, a sender and a receiver should previously agree on some set of —possibly implicit— conventions in order to (a) identify a location in their respective data repositories; (b) identify the data item to be incorporated. Thus, a communication event could be described as

$$\# \rightarrow \# \, \alpha \, b \qquad (1)$$

where # is a specific data item in a data repository, α is an assembling instruction, and *b* is a data item to be attached. The arrow denotes the transition between the two states in a data repository. A simple example of a communication event is a binary message, where # is the last bit received, *b* is the bit to be assembled, and α is an instruction to assemble *b* to # by using the one-dimensional adjacency relation.

As a further example, if *H0* is a binary string representing a specific horizontal line on a display and β is the adjacency relation between a line and the one immediately above, then the following expression would describe the assembling of a new line, represented as *H1*, on top of the extant *H0*, i.e.,

$$H0 \rightarrow H0 \, \beta \, H1$$

In yet another example, if *T* represents a specific triangle and *Z* represents the shape of a zebra, then a receiver could implement the information event

$$T \rightarrow T \, \gamma \, Z$$

where the symbol γ indicates that the shape *Z* is to be embedded in the shape *T*. The information conveyed in this example would be unacceptably vague for an engineer, but in most practical situations people are usually content to learn that a frog is inside a pond, and will not ask for the precise coordinates of the frog.

A fourth example involves a house as made up of identifiable parts. The expression

$$roof \rightarrow roof \, \kappa \, chimney$$

would describe the assembling of a chimney on top of the house's roof by means of the three-dimensional adjacency relation κ. Again, this kind of information is largely imprecise, yet fairly acceptable in practice.

The actual identification in a data repository of the location # in (1) would require the existence of a referencing system shared by both the sender and the receiver. A referencing system could be implemented by introducing a structure in a data repository. In the following section, a possible such structure will be defined.

III. DATA AGGREGATES

Data items could be arranged in a variety of ways, including representational (e.g., labels on a map), encoded (e.g., data streams), physical (e.g., material items in drawers), etc. From an abstract standpoint, any such arrangement could be described as a number of symbol aggregates endowed with a particular structure in the form of tables, graphs, objects or other ways of organization [28][14]. Natural Language (NL), being a means to deliver information, could also be argued to use data but, except in specific, explicitly structured subject areas, its users are usually unaware of the structure of such data. Describing a data structure consistent with NLs would therefore be a first step to characterize the nature of language as a tool to convey information.

For a general approach to a diversity of data arrangements, the term 'data aggregate' will be adopted here. A data aggregate is defined as a number of data items that could be represented as points on a surface. A data aggregate is arguably the minimal structure that can be conceived of, and it does not exclude other additional

structures. Thus, a number of colors could be considered as a data aggregate, irrespective of whether they can be synoptically represented as points across a rainbow or on a painter's palette. In a data aggregate, items can be pointed to but do not have to be distinctly labelled —you may know nearly everything about a wood and not have a name for any of its trees—. Also, a data aggregate could be indefinitely updated, as long as any new item could be represented as an additional point on the same surface. Goats in a herd, or symbols on a paper sheet, are simple examples of data aggregates.

Items in a data aggregate could be discriminated by applying criteria to them. A criterion is a notion more general than a property, because it encompasses properties as well as fancy choices and algorithms. Two of the simplest criteria that could be applied to a data aggregate are the ones associated to intension and extension. Aggregations of items in a data aggregate do not have any extension or intension connotation *per se*, and are therefore objects more general than sets. Extension and intension could be implemented on them by means of resp. the logical connectives $\wedge$, $\vee$, e.g.,

blue $\wedge$ red $\wedge$ yellow $\wedge$ ... extension [colors]
blue $\vee$ red $\vee$ yellow $\vee$ ... intension [color]

In general, therefore, a number of items u_1, u_2, u_3, ... in a data aggregate could be discriminated in two alternate ways, i.e.,

$u_1 \wedge u_2 \wedge u_3 \wedge ...$ (2)
$u_1 \vee u_2 \vee u_3 \vee ...$ (3)

Because mathematical sets are said to be describable both in extensional and intensional terms, we shall rather not mix things up and separately discern either description instead. Hence, any expression in the form (2) will be referred to as a **combination**, while any expression in the form (3) will be referred to as a **category**. This difference reflects the use of plural resp. singular in human language. Thus, 'colors' could be associated to a combination, while 'color' could be associated to a category. When the scope of the criteria is not specified, the expressions (2) and (3) could also be interpreted as reflecting the difference between resp. 'every' and 'any'.

Any item *u* encompassed by a category *C* —i.e., complying with the criteria that define *C*— will be referred to as an **instance** of *C*. A category *C* encompassing the instances u_1, u_2, u_3, ... will therefore be expressed as

$C \equiv u_1 \vee u_2 \vee u_3 \vee ...$

The definitions of category and instance could be used as a means to locally refer to an item in a data aggregate. Indeed, in a data aggregate *E* where a category *C* has been identified, any instance of *C* could be expressed as a disambiguation of *C*. That is, if we denote a category as *C()* and an instance *u* of that category as *C(u)*, we could refer to *u* as

$C() \rightarrow C(u)$

The expression above may be interpreted as a path in *E*, i.e., "select *C*, then select the instance *u* of *C*", where *u* could be identified by means of either a label or a number of instructions. In the following sections, an enlarged notion of disambiguation will be shown to be a powerful device to refer to data items in a data aggregate —arguably, the basic addressing device used by natural language users—.

A data item in a data aggregate could also be referred to through a disambiguation of a number of categories it might be ascribed to. For example, the word 'green' may denote either a color or a political affiliation. In the absence of any additional cues, they could be disambiguated resp. as either *color(green)* or *political_affiliation(green)*. In formal terms, if *M* is a category having the category *C* as an instance, then the instance *C(u)* could be referred to as

$M(u) \rightarrow C(u)$

The notation used thus far, based on symbols such as connectives or arrows, will be referred to as **symbol syntax**. An alternative syntax, which shall be referred to as **string syntax**, would express categories and instances as single words, and disambiguations as strings, as follows:

symbol syntax	string syntax
C()	C
$C() \rightarrow C(u)$	C [δ u]

where the symbol δ denotes the relation linking a category with any of its instances, i.e., the fact that *u* complies with the criterion that defines *C*. The correspondence between symbol expressions and string expressions will be denoted as >>, e.g.,

color() >> color
color() $\rightarrow$ color(blue) >> color [δ blue]

Note that, in practice, if we deem it obvious that, e.g., the word 'blue' refers to a color, we will not precede it with the word 'color', which will have to be guessed by the receiver. This data compression feature reflects an implicit operation that pervades human language —and arguably also human thought—, i.e., spontaneous categorization. The feature of spontaneous categorization will be further elaborated in Section VII.

IV. COMBINED CATEGORIES AND CATEGORY CLUSTERS

The connective $\wedge$ could also be used to discriminate combinations of categories in a data aggregate. For example, from the categories

mass, electric_charge, spin

a combined category could be derived, which in turn would give rise to a number of objects, e.g.,

mass ∧ electric_charge ∧ spin >> particle
mass ∨ electric charge ∨ spin >> observable
mass(9.1×10^{-31} kg) ∧ electric_charge(-1.6×10^{-19} C) ∧ spin(1/2) >>
particle [δ electron]

As the latter example shows, a combination of categories is itself a category, having as instances combinations of instances of its component categories. The latter example unequivocally describes the item 'electron' as a full disambiguation of the category 'particle'. However, combined categories could also be partially disambiguated by specifying just some instances of its component categories, e.g.,

bearing() ∧ altitude(7000 ft.) ∧ No._of_passengers(80)

Component categories in a combined category could also be discriminated by means of the connective ∨. The resulting object could be used to identify a path within the data aggregate individually leading to them. For example, the category 'color' could also be construed as an attribute, i.e., it could be referred to as an instance of the category

color ∨ shape ∨ size ∨ ...

Categories pervade language, a fact which is obscured by the spontaneous categorization mechanism, of which language users are usually unaware. In human languages, spontaneous categorization is a run-of-the-mill feature, associated not only to adjectives such as 'blue' or 'big', but to virtually any kind of meaningful component. Thus, the sentence 'birds fly' could be misleadingly categorized as implying that hens fly, or meaningfully categorized by instead interpreting 'birds' as an instance of some category, e.g., birds ∨ mosquitos ∨ bats ∨ ... having 'fly' as an attribute. Similarly, the meaning of 'a through person' could only be captured by evoking a category of concepts having 'through' as an instance.

In general terms, a **combined category** G^{η} will be defined as the general expression

$$G^{\eta} = O1 \wedge O2 \wedge O3 \wedge ... \quad (4)$$

where *O1*, *O2*, *O3*, ... are categories, whether they have been disambiguated or not, and η uniquely identifies that particular combination of categories. The definition (4) is consistent with a number of concept theories [45][47], that describe concepts as n-tuples of symbols representing attributes.

An n-tuple is just a one-dimensional combination of categories, and therefore a particular case of the more general concept of **category cluster**, where complex spatial relations could be incorporated as additional discrimination criteria in a data aggregate. A data form is a familiar example of category cluster. In general terms, therefore, a **representation** can now be defined as a data aggregate together with any number of category clusters.

V. Reference and Updating within a Category Cluster

Given a category cluster *G* and one of its component categories *C*, any set of instructions *r* to uniquely identify *C* within *G* will be referred to as a **relation** *r(G, C)*. As in the one-dimensional case, specific category clusters could also be referred to by specifying one or more of its component categories. For example, an employee's record might be uniquely identified by specifying just the employee's name, or his age and height. This could be formally described as follows. Let *G* be a category cluster, *r* a set of instructions to identify *C* within *G*, and G_k a copy of *G* where the data item *u* has been specified for the category *C*. The category cluster G_k could therefore be referred to as

$$G_k = G \mid r(G, C(u)) \quad (5)$$

i.e., G_k can be interpreted as a partial disambiguation of *G*. In string syntax, this will be expressed as

G | r(G, C(u)) >> G [r u]

For example,

ball = shape(round) ∧ color() ∧ size()
$ball_k$ = shape(round) ∧ color(red) ∧ size(big) >> ball
[r_2 red] [r_3 big]

where r_1, r_2, r_3 would represent resp. the sets of instructions to locally identify each of the component categories 'shape', 'color', 'size'. If a reference would not result in a full disambiguation, further disambiguating [r u] legs could be appended until a unique reference is achieved. In the general case, therefore, the disambiguation of a category cluster *G* will be expressed in string syntax as

$$G\ \Sigma[r_j\ u_j] \quad (6)$$

where Σ denotes a string made up of $[r_j\ u_j]$ pairs, u_j denotes an instance of the component category C_j, and r_j denotes the relation $r_j(G, C_j)$. The possibility to uniquely identify a category cluster even when only some of its component categories have been specified is a feature heavily used by natural language users as a data compression device. Indeed, if there is only one red ball in the room, you would hardly want to refer to it as "the big red expensive air-filled ball on the sofa".

Any set of rules to convert string syntax expressions into different strings will be referred to as a **conventional** syntax. For example,

String syntax	*Conventional syntax*	
ball [r_2 red] [r_3 big]	big red ball	(English)
boule [r_2 rouge]	boule rouge	(French)
bam [r ug]	bugam	(imaginary)
✌ [r_2 👌] [r_3 🖐]	👌✌〰🖐	(non-word)

The notion of category cluster, plus the relations it entails, endow data aggregates with powerful structures that could be used to semantically represent a vast number of concepts, e.g., ontologies, verbs, or semantic representations of space-time concepts, together with a local mechanism to refer to them [37][1][40]. A few basic semantic clusters will be described in Section XII.

The definition of the cluster-category relation also implies that categories could be referred to in terms of the cluster or clusters they are part of. This stems from the definition of the converse relation. Given a relation *r(G, C)*, the expression

$$C \rightarrow C \mid r(G, C) \quad (7)$$

describes the constriction of the general category *C* to the range of instances allowed in *G*. In string syntax, (7) will be expressed as

$$C\ [r'\ G] \quad (8)$$

and *r'* will be referred to as the **converse relation** of *r*. Example:

$$color \mid r_2(ball, color) >> color\ [r'_2\ ball]$$

If we now substitute (6) into (8), the general expression will therefore be

$$C\ [r'\ G\ \Sigma[r_j\ u_j]] \quad (9)$$

When $G\ \Sigma[r_j\ u_j]$ describes a full disambiguation, the expression (9) will point to the unique content of *C* in *G*, i.e., it could be used to indirectly refer to a specific component instance in a category cluster. For example, the expression

$$color\ [r'_2\ ball\ [r_1\ big]\ [r_{on}\ sofa]]$$

might uniquely refer to the color of the ball on the sofa without directly using its name. The expressions (6) and (9) may be seen as a formalization of the Fregean concept of sense (Sinn) [21], which would interpret the notions of Sinn and Bedeutung as part of a wider picture. Thus, in Fregean terms the category-instance expression C(John) would be the meaning of 'John', while the indirect references 'the tall man' or 'the man with my hat' could be used to express two of the multiple possible senses of the referent 'John'.

The expressions (6) and (9), used to refer to resp. category clusters and component categories, could be used not only to refer to data items in a structured data aggregate, but a sender could also used them to update the receiver's presumed representation. For example, if the receiver is believed to ignore that there is a ball on the sofa, then that information could be sent by means of (6), i.e.,

$$!ball\ [r_4\ sofa]$$

where the symbol ***!*** denotes a new category cluster to be included in the receiver's representation. Such updating is a commonplace device used in natural language exchanges, to indicate, e.g., that a new character has appeared in a film, or a new guest has arrived at a party. If the receiver were presumed to know that there is a ball on the sofa but not its size, then that information could be conveyed by means of the expression

$$ball\ [r_4\ sofa]\ [r_3\ big!]$$

where ***!*** now denotes an instance intended to fill a category presumed to be empty at the receiver. In a converse situation, where the sender ignores some information item supposedly known by the receiver, the expressions (6) and (9) could also be used for querying purposes, by pointing to the required item by means of a different symbol, e.g.,

$$?\ [r_4\ sofa]$$
$$?\ size\ [r'_3\ ball\ [r_4\ sofa]]$$

where the symbol ***?*** points to resp. an category or instance unknown by the sender. The referencing and active/passive updating uses of (6) and (9) could be summed up as follows

reference	G [r u], C [r' G]	(10)
updating	!G [r u], G [r !u]	(11)
querying	? [r u], ?C [r' G]	(12)

Depending on the rules devised to derive specific syntaxes from the general expressions (6) and (9), a large number of unfamiliar grammars could be built, whether or not in use by any communities of users. This should make it possible to test the validity of the approach developed above. Indeed, that validity would be challenged if some grammar in use were found whose syntax rules could not be derived from (6) and (9). Conversely, a weak confirmation could be obtained by checking whether a number of 'exotic' syntaxes could be derived from (6) and (9). Amazonian pirahã [18] and Australian warlpiri [35], among others, would seem to be good candidates [17].

VI. PROPOSAL FOR A PARSER

Based on the general expressions (6) and (9), a variety of parsers from conventional syntaxes into string syntax could be devised by expressing lexical/morphological components in terms of categories, instances, and relations. Although any such component is potentially susceptible to be categorized —the atypical syntax of 'a through person', mentioned above, provides a telling example—, a basic list of usual category/instance values could be established as follows:

adjective	u
preposition	r
noun	G
verb	G [r u]

where *u* denotes an instance, *G* denotes a category cluster (or, in the simplest cases, a category), and *r* denotes a relation. Note that verbs have been expressed as tense-carrying items, where *G* denotes the verb root and the *u* in [r u] denotes an instance of the tense category 'past/present/future/...' Based on the list above, qualifying pairs, such as 'big ball', 'car wheel', 'take book', or 'run fast' would be expressed as disambiguations in the form G [r u]. Therefore, as a first step a number of string segment types could be identified, i.e.,

SL *noun* $\Sigma[r_j\ u_j]$ *verb*
SR *verb* $\Sigma[r_j\ u_j]$
W *noun* $\Sigma[r_j\ u_j]$
M *preposition noun* $\Sigma[r_j\ u_j]$
A *adjective* $\Sigma[r_j\ u_j]$
v *verb* [r *tense*]
n *noun*/*verb root*
α *adjective*

where each individual leg [r u] would express a qualification of the preceding component. A few English examples may illustrate this, i.e.,

SL *birds* [] *fly*, *a book* [on the shelf] *fell*
SR *ran*, *saw* [her] [with a telescope]
W *the book* [from the shelf], *happiness* []
M *under* the [milk] *wood*
A [nearly] *perfect*
v [would] *hope*
n *wheel*, *pampering*
α *big*, *unthinkable*

Based on such components, the following prolog-like basic rules could be stated, together with their output:

is(x v, SL) :-	is(x, W)	→ v [b x]	*1
is(v x, SR) :-	is(x, W)	→ v [b x]	*2
is(v x, SR) :-	is(x, W M)	→ v [b x] M	*3
is(v x, SR) :-	is(x, W M M)	→ v [b x] M M	*4
is(x, A) :-	is(x, α)	→ α	*5
is(x n, W) :-	is(x, A)	→ n [b x]	*6
is(x n, W) :-	is(x, n)	→ n [b x]	*7
is(x b y, W M) :-	is(x, W), is(y, W)	→ x [b y]	*8
is(x r y, W) :-	is(x, W), is(y, SR)	→ x [r y]	*9
is(x r y, W) :-	is(z, W), is(y, SL)	→ x [r y]	*10
is(b x, M) :-	is(x, W)	→ [b x]	*11

Example: S ≡ the book on the shelf fell

is(S, SL) :- is(book on shelf, W) → fell [b book on shelf]	*2
is(book on shelf, W) :- is(book on, α) → fail!	*5
is(book on shelf, W) :- is(book on, A) → fail!	*6
is(book on shelf, W) :- is(book on, n) → fail!	*7
is(book on shelf, W) :- is(book, W), is(shelf, W) → book [on shelf]	*8

Output: fell [b book [on shelf]]

An educated guess based on a number of partial implementations by the author suggests that a few hundred rules would probably suffice to process most sentence types.

VII. SPONTANEOUS CATEGORIZATION

Parsing is a way to convert NL strings into combinations of (6) and (9), but cannot always be used to decide whether such expressions are to be interpreted as (10), (11) or (12), i.e., whether they are intended as reference, updating or querying. Querying purposes are usually denoted in various ways, including characteristic sentence structures, question marks, and/or prosodic patterns, but updating purposes (i.e., predication) may not always be obvious, at least in written form. For example, in Maya language the written expression *keel winik* can be interpreted as either *the man is cold* or *the cold man* [43]. In such cases, deciding whether a NL message is meant as (8) or (9) should be the job of spontaneous categorization.

Spontaneous categorization has been identified and studied from the standpoint of language use [15][40], but also in children [26][7][32], nonhuman primates [24] and even distantly related species [25]. Interestingly, a comment in Shannon's seminal paper [41, op. cit.] hints at the role of categories in NLs: "The significant aspect [of communication] is that the actual message is one *selected* from a set of possible messages" [emphasis added].

In the binary case, the set of possible messages is easy to determine, since it can be derived from the category 0/1. The information conveyed by Beethoven's Ninth Symphony, though, may be harder to determine, because one category of possible messages is the category of all possible symphonies. Fortunately, however, some partial information can be extracted from it. For example, we can determine that it consists of four movements, what kind of movements they are, how many instruments are playing it and, ultimately, each of its notes.

NL strings can similarly be decomposed in different ways, which is particularly apparent from the use of questions. As an example, the questions

who left at eight?
when did Joan leave?
what did Joan do at eight?

refer to different categories, i.e., person, time, and action, all of them implicit in the expression 'Joan left at eight'. Thus, depending on the part of the message that may be selected and the category inferred from it, one single expression could be used to refer to different information items. In most cases, the categories would be implicit, and its identification would be left to the receiver through a spontaneous categorization of the message received.

The following example might help to clarify this. Suppose that you are at home, sitting in front of your TV screen, when suddenly the telephone rings. You answer the call. It's your friend Zoe.

"Hi, Zoe. No, I didn't feel like going out tonight. I'm watching a film."

It is 9:15 PM on a Monday, so your friend is not surprised. Now imagine that you and her have instead decided to go to the movies that day so that, at some point, the two of you are watching a film together. Suddenly, you move closer to Zoe and whisper:

"I'm watching a film."

Of course, this is no news for Zoe, who, puzzled at first, faces subsequently a critical decision: either you have become mad, or you were meaning something. Apparently, your statement has not provided her with any information. She already knows that you are watching a film. Her mind works frantically. What could you have possibly meant?

Zoe comes up with a number of possibilities. Perhaps it has been a long time since you last watched a film, and you are just expressing joy. Or you might happen to work in the archives of a film library and are usually busy handling films without ever getting to watch any of them. Or perhaps you are a fanatic of theatre and tonight, exceptionally, have condescended to go to the movies. Thus, what Zoe's mind would be doing is to construct a background against which to extract information from your message. As she might find out, any of the following constructions would be apt to provide plausible information from your message:

"For a long time, I *have not watched* a film. Now I *am watching* a film"
"Usually, I *handle* films. Now I *am watching* a film"
"Usually, I watch *theatre plays*. Now I am watching *a film*"

Each of these interpretations conveys the information that something is happening that did not use to happen. To extract information from your message, Zoe has had to mentally construct resp. the categories

not watch / watch
handle / watch / ...
a play / a film / ...

In more abstract terms, she has constructed the ambiguous messages:

I am X
I am X a film
I am watching X

and, based on a single instance taken from your message, has subsequently let the categories X spontaneously form in her mind. The information she will eventually extract from your message will depend on what those categories actually encompass and how their contents fit into the information she already has about you. The ability to infer a category from one of its instances, i.e.,

b ⋯> C(b)

is what has been referred to as spontaneous categorization, and would thus seem to be a prerequisite to process a NL message, and possibly a distinctive feature of human brains. It should be noted that some categories may not have a name themselves, thereby exposing a lexical gap. Thus, 'green', 'red' and 'blue' could be ascribed the category 'color', but there is no single word in English for the category that encompasses the states green, ripe and intermediate ones as applied to a fruit (although the derived noun 'ripeness' is sometimes remedially used, as in 'degree of ripeness').

If you and Zoe had instead talked on the phone, the process would have been simpler but, essentially, not different. Upon hearing your reply, Your friend Zoe would simply have evoked a category of actions to be expected from you in your place on a Monday at 9:15 PM, i.e.,

I am X

where X = eating / sleeping / reading / watching a film / ... From the moment she had you on the phone, Zoe was predisposed to wonder what you might be doing at the other end of the line, and the words "watching a film" would be a good answer to that. However, in the presence of an obvious context, like the film the two of you were watching, the effort required from Zoe is paradoxically far more demanding. The key to Zoe's clairvoyance on the phone is that she had a much smaller number of choices as to what your message could be referring to.

Depending on the communication context, spontaneous categorization makes it even possible to dispense with grammar rules. Spontaneous categorization is what enables us to understand ill-formed expressions in colloquial utterances, or due to some lapsus linguae, or uttered by a foreigner with a non-standard syntax. For example, a sentence assembled with the words "place", "cheap", "eat" and pronounced by a likely hungry foreigner could be easily interpreted as

?place [r_1 eat [r_2 cheap]]

VIII. CODE VS. MEANING

In Section IV it has been shown that category clusters can be used to generate syntax. However, neither the nature of categories, nor of the relations that 'glue' them into category clusters, have been addressed. Thus stated, the task looks to be huge, but might be at least partially manageable. While the structure of a data form, or a database table, is usually designed for convenience, ontologies and other spatial layouts tend to be representational, i.e., are intended to express meaning in a more direct way. Therefore, it may be worth exploring to what extent category clusters could be used to directly express meaning.

Parsing —and, more generally, message decoding— is a rather convoluted way to extract meaning. It is based on the capability to identify groups of symbols connected to each other by the one-dimensional spatial adjacency relation. The actual relations that those symbols are intended to reflect are

only accessible through a number of conventions on how to group them up and translate them into meaningful representations. However, symbol strings could also be used to directly express meaning, i.e., as representations themselves. For example, the string

Su Mo Tu We Th Fr Sa (13)

is intended to directly represent temporal relations, i.e., it reflects a semantic intent, and to that effect the way it has been constructed —and therefore its information content as a message— is of little relevance. Ironically enough, whereas for Shannon's purposes the semantic aspects of a symbol string are irrelevant, for semantic purposes it is the information content of a symbol string which is irrelevant. In representational terms, the string (13) is a category cluster and, as such, it already reflects a key advantage of the representational approach, i.e., unlike messages, which are updated through the rigid process of streaming, category clusters can be updated locally. A simple illustration of this potential can be derived from binary strings.

A binary string can be seen as a category cluster consisting of a single category, i.e., 0/1, and a single relation, i.e., the spatial adjacency relation linking each binary digit to the next one. A binary information event could be described by the general expression (1), where # denotes an existing binary string, *b* denotes a bit to be assembled, and α denotes the one-dimensional adjacency relation between any two consecutive bits. The location where each new bit is to be assembled is referred to by means of a meta-reference, i.e., 'the last bit received'. However, if the final purpose is to reconstruct a bit string rather than assemble it in a sequential way, it might be more convenient to make use of local referents. Once we have put the chimney in place, we may want to install the front door, and to do that the reference to the last item installed would be useless.

In a bit string, a simple way to refer to a local component is to identify a unique feature in the string. For example, let *M* be the following message:

1 1 0 1 0 1 1 1 0 1

The string above includes a number of unique substrings, e.g.,

1 1 1
1 0 1 1
1 1 1 0 1

The sender could not refer to the last bit received as '1', which would be ambiguous at the receiver, but it could refer to the unique substring '1 1 1 0 1' instead, and instruct the receiver to append the bit *b* by means of the adjacency relation α, i.e.,

1 1 1 0 1 → 1 1 1 0 1 α b

Thus, by using unique substrings as referents, the sender could choose to instruct the receiver to construct just parts of *M*, instead of sending the whole *M*. For example:

1 0 1 1 → 1 0 1 1 α 1
1 0 1 0 → 1 α 1 0 1 0

This would be a local approach, insofar as each event would describe only a partial reconstruction of *M*. It may not sound very appealing as an alternative to the good old 'next-after-last' convention. But if our source were a two-dimensional matrix instead of a one-dimensional string, things would start to look different. For example, on a display described by a digital matrix *M*, any solid image of an object will be described by a subarray of *M*, and the general expression (1) could be used to describe the assembling of a subarray *u* to the subarray # by means of the two-dimensional adjacency relation *r*. Thus, if # represents the image of a rug and *u* represents the image of a ball, then a mutually agreed definition of *r* would make it unnecessary to transmit a whole matrix *M* describing a room just to indicate that there is a ball on the rug. Such local approach dispenses with the need to describe any parts of a data aggregate that the sender deems irrelevant. It may entail the use of 'vague' descriptions, as with 'the frog in the pond'. However, the gain in nimbleness is huge.

IX. DENOTATIONAL COMPRESSION

The mention of 'vague' descriptions points to an additional feature of representations, i.e., the fact that they can compress information by making use of objective relations such as the spatial adjacency relation. Not unreasonably, the human mind tends to simplify things. If you live in Paris, you will probably be able to lay out a mental map of the town and then locate the Mona Lisa on it. But if you have always lived in an Amazonian tribe, isolated from the external world, and one day you hear about Nessie, described to you as some unspecified monster that lives in some faraway lake, could you still be expected to mentally represent Nessie? Most likely, your representation will hardly be good enough for you to reach Scotland. But, even so, you might be capable of mentally representing the Earth as a sphere (maybe simply as a plane!) having an island somewhere on its surface, in the island a lake, and in the lake an animal whose shape you cannot tell. As long as you do not assign a specific shape to the island, the lake, and the monster, your representation of them would somehow 'float' in some vague —but certainly structured— territory of your mind.

This seems to suggest that the human mind makes use of preexisting configurations, or 'structures', to lay out the information derived from perceptions, a topic that has already been addressed from different perspectives [19][4]. Short of additional data, we are often content to know that an island is a surface inside another surface, or that Nessie is some three-dimensional volume inside a volume. Or, most simply perhaps, a dot somewhere on a surface. The point here is that a perceptual representation is alright as long as it is made up of territories and borders and these do not break

up. Subject to those rules, perceptual concepts are largely malleable.

This is probably not by chance. Imprecise representations happen to be a good way to compress information. For example, the image of a lake surrounded by land could be roughly emulated by means of 0s and 1s:

```
... 0 0 0 0 0 ...
... 0 0 1 1 1 1 0 0 ...
... 0 0 1 1 1 1 0 0 ...
... 0 0 0 0 0 ...
```

that is, as an 'area' made up of adjacent 1s (water) and surrounded by an 'area' made up of adjacent 0s (land). Now, denotationally speaking it would have sufficed to write:

```
0
0 1 0
0
```

where the symbol 1 represents a connected surface, and the four 0s represent a surrounding open surface externally adjacent to it. If needed, the compressed figure above could be expanded into the uncompressed one by 'unfolding' (i.e., replicating) the 1s and 0s as wished, in all directions, as long as the adjacency relations do not get breached. This is to say that the 1s and 0s would unfold into themselves, and an unfolding such as

```
... 0 0 0 ...  -> ... 0 1 0 ...
```

would be a violation.

Actually, the representation above could be as useful to denote a lake as an island —or, for that matter, a stain of spaghetti sauce on someone's shirt—. From such abstract representation, specific denotations could be derived by associating to the 0s and 1s the symbols 'water' and 'land', or 'land' and 'water', as the case may be. The underlying structure, however —that is, the conglomerate of adjacency relations— will stay unchanged.

The implication is that there exist denotational representations that are independent from the denoted object. Such representations may or may not be maximally compressed. If it were not possible to further compress one such representation, then we would be in the presence of an irreducible concept —i.e., a semantic primitive—. Certainly, it is not possible to compress a line into a dot, a tree into a line, etc. without altering their adjacency relations.

This would suggest that the structures human brains use to 'interpret' perceptions stem from a more abstract reality. Might such structures be a repertoire of topological configurations? That would be really good news. If the meaning of words were ultimately irreducible denotational configurations, human language would be based on an absolute referent: geometry. Meanings could be systematized, studied and compared according to objective criteria. Concepts could be handled by means of symbols and, consequently, would be computer-processable.

X. CATEGORY STRUCTURES

Spontaneous categorization has been described as an operation whereby categories are evoked to accommodate symbols received, presumably in such a way that those symbols could be subsequently retrieved. Therefore, it seems legitimate to wonder if such categories have some identifiable structure that makes them readily accessible for both storage and retrieval. Furthermore, our neuronal system cannot be impervious to the reality that space can be decomposed into three dimensions, that colors form a continuum, or that musical notes sound one after another.

The role of space and time structures in perceptual and cognitive processes has been extensively addressed in the literature from both the theoretical and experimental standpoints [22][50]. The precise relation between perceptual and cognitive structures is still being delved into [10], but the recent encompassing notion of cognitive architecture [36] opens the door to a functional approach to cognitive processes that is rather independent from their biological basis.

Therefore, if it would be possible to formally characterize category structures in terms of spatial and temporal relations, such relations would naturally give rise to objective category clusters of a semantic nature, and hence to at least some building blocks of meaning.

In fact, a close look at categories does reveal substrates that are intrinsically different in nature, and where the spatial adjacency relation plays a key role. In the preceding Section, the role of adjacency relations in spatial representations has already been noted. As it happens, a number of perceptual categories could also be shown to reflect underlying structures involving adjacency relations. Consider, for example, the different categorizations the undefined word *splack* elicits in the following sentences:

The sauce tasted too splack, so I added some garlic

I waited for you from noon to splack

The stripes blue, splack and yellow on her skirt

Bugs Bunny turned into a splack, then into a donkey

He pointed to the splack of the triangle

The categories that could be evoked from the examples above entail different underlying structures, i.e., resp.

- A number of individual, fuzzily connected concepts (tastes)
- A 1D continuum of concepts spanning from noon to splack (time values)
- A 1D, rainbow-like continuum of concepts (colors)
- A continuum of concepts spanning from splack to donkey (shapes)
- A finite number of spatial features (vertices/sides/areas)

The structures thus evoked are characteristically different in nature. The taste 'salty' could become 'sweet'

by progressively adding salt and removing sugar, and a similar transition could be devised between any two other tastes. Intermediate tastes, however, are fuzzy, i.e., there seems to be no referent to precisely locate them with relation to others. No matter how many new intermediate tastes we identify, the structure will reappear, and any two tastes will always be separated by a conceptually blurred territory.

Time events do not have the same properties as tastes. We tend to think that an event is farther in time the more events can be piled up between it and now, but we cannot foresee how soon the next shooting star will show up in the sky. However, we can locate along time —e.g., on a clock's face, or in a calendar— any event that happens between two other identifiable events. And we can do it unequivocally, i.e., we cannot conceive of two different time paths connecting one event to another.

Color ranges are categories familiar to those who work with spectrometers, or check a catalog in a cloth shop. Color ranges have a one-dimensional structure, and share a number of features with time spans. But, unlike time, which is unlimited towards both the future and the past, a color wheel can be constructed that closes upon itself, and therefore colors are consistent with a circular structure.

Unlike time or color, the category that encompasses Bugs Bunny's and other intermediate shapes is two-dimensional. Bugs' shape could turn into almost any two-dimensional shape, and there are uncountable ways for it to turn into a donkey. While we can mentally 'travel' from noon to splack to midnight in the same way as a point would travel along a one-dimensional line, Bugs Bunny's shape could evolve through an infinite number of shapes before appearing as a splack. Even so, that evolution could be decomposed into three different steps. During a first stage, the intermediate shapes between Bugs and e.g., a donkey will remind us of Bugs for a while, then will seem unrecognizable for some time and, in the final stage, will remind us ever more vividly a donkey's shape. In representational terms, those three stages are not unlike transitions between tastes or colors.

Lastly, in the 'splack of the triangle' example, any categories that could be evoked from a triangle are inherently finite, and no two of them will be connected by a fuzzy territory. A vertex is immediately adjacent to two sides, a side is immediately adjacent to two vertices, and the inner and outer area of the triangle are immediately adjacent to the triangle's contour and any of its components. Unlike tastes, time values, colors, or shapes, which could be indefinitely updated, none of those categories could possibly be updated and still be thought of as making up a triangle.

Similarly, a data form made up of delineated cells is usually laid out as a two-dimensional structure. As long as its cells' borders are kept from merging or breaking, its structure could be deformed at will without essentially altering its functionality. In other words, what characterizes a data form with relation to other conceivable forms is, precisely, its specific adjacency relations.

All of these structures and their properties strongly remind of a branch of Mathematics known as Topology. With a few intriguing differences, though. Broadly speaking, Topology characterizes objects based on their potential to transform into each other without breaking or merging at any point. It conceptually groups geometric objects in terms of features such as the lines, surfaces or volumes they are adjacent to.

However, open-sets Topology differs from denotational topology in a few respects. Thus, if a circle gets broken at one of its points, the breaking point can only stay adjacent to one, not both, of the free ends, and the resulting object will be a stretch of line lacking one accumulation point. From a cognitive perspective, though, both ends will be equally denotable, irrespective of the fact that one of the ends is, in mathematical terms, a missing accumulation point. Conversely, for a segment to be made into a circle the names of its two ends would have to merge into one, thereby dropping one denoting symbol. In sum, for denotational purposes whenever you change the adjacency you have to add or remove labels —i.e., switch to a different *denotational* structure—.

Topology, on the contrary, is not concerned with labels. Topology is about open sets, not representations. Therefore, we will not be concerned here with the mathematical continuity of a representation, but with whether and how a spatial configuration can be consistently denoted (i.e., structurally identified and labelled), as well as with the characteristic properties inherent to it. This is to say that the human mind compresses sensory information by retaining only discontinuities, i.e., the adjacency relations between regions of different dimensions (points to lines/surfaces, lines to surfaces/volumes, etc.). Which makes sense, because where there is no discontinuity there is nothing to denote. As was the case with the unnamed trees in a wood, denotability does not mean that a symbol should be automatically attached to each identifiable feature, but only that these can be used as symbol holders, whether we decide to 'fill' them or not.

The characteristic structures that the spatial adjacency relation gives rise to could be formalized by introducing the concept of *denotational jigsaw*.

XI. Definition of a Denotational Jigsaw

A **denotational jigsaw** is defined as a number of objects z_1, z_2, z_3, ... together with a number of adjacency relations $k_{mn}(z_p, z_q)$, where m is the dimension of z_p, n is the dimension of z_q, and $m \neq n$, so that for any two objects u, v in the jigsaw there is always a 'path' connecting u to v, i.e., in a simplified notation:

$$k(u, w)\ k'(w, b)\ k''(b, c)\ ...\ k^{(j)}(y, v)$$

In this simplified notation, the symbols k, k', ... $k^{(j)}$ denote the corresponding adjacency relations. Note that an adjacency relation is symmetrical:

$$k_{mn}(u, w) = k_{nm}(w, u)$$

but not identical nor transitive:

$k_{mm}(u, u) \Rightarrow m \neq m$
$k_{mn}(u, w), k_{nr}(w, z) \not\Rightarrow k_{mr}(u, z)$

Thus, a surface containing a circle B_1 with an inner area B_2 and an outer area B_3 could be described as the jigsaw (Figure 1).

B_1, B_2, B_3

$k_{21}(B_2, B_1), k_{12}(B_1, B_3)$

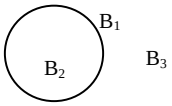


Figure 1. Circle-within-an-area jigsaw

In this configuration, the inner area B_2 is adjacent to the circle B_1, but not to the outer area B_3, and no adjacency relation between B_2 and B_3 can be transitively inferred from $k_{21}(B_2, B_1), k_{12}(B_1, B_3)$.

The identification of a point B_4 within B_1 would transform the circle B_1 into a closed segment, thereby creating a more complex set of adjacency relations (Figure 2).

B_1, B_2, B_3, B_4

$k_{21}(B_2, B_1), k_{12}(B_1, B_3)$
$k_{02}(B_4, B_2), k_{02}(B_4, B_3)$
$k_{01}(B_4, B_1)$

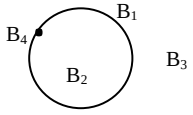


Figure 2. Point-within-a-circle-within-an-area jigsaw

For different values of m, n, the following simple paths can be identified:

k_{10}(line, point)
k_{20}(area, point)
k_{30}(volume, point)
k_{21}(area, line)
k_{31}(volume, line)
k_{32}(volume, area)

which could be assembled into more complex ones, e.g., a triangle, a polyhedron, or a category cluster. Insofar as dimensions are themselves irreducible, the above paths are irreducible, and are not incompatible with each other.

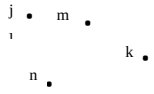
Some of those complex configurations, e.g., a triangle, can be described in terms of a finite number of adjacency relations. Unlike finite configurations, however, towns on a map, or events along time, are open configurations, i.e., they can be indefinitely updated. New elements could be added to them without altering its denotational nature, and therefore they can be described as a self-similar structure, based on a finite number of updating rules.

For example, if we denote as α the adjacency relation k_{30}(v, i) between a point *i* and its surrounding volume *v*, the same relation α could be used to describe any number of additional points *j*, *k*, ... within that same volume, i.e.,

$v \alpha i, v \alpha j, v \alpha k, ...$

and the identification of a new point n could be described by means of the rule

$v\{...\} \rightarrow v\{... n\}$ (S0)



where

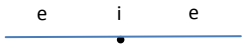
$v\{i, j, k, ...\} \equiv v \alpha i, v \alpha j, v \alpha k, ...$

Events along time, or towns along a railway, however, fail to be describable by means of the updating rule (S0), and their description requires a different set of rules. If we denote as β the adjacency relation k_{10}(line, point), any two points *i*, *i* will be adjacent to a one-dimensional stretch *e* separating them, i.e.,

$i \beta e \beta i$

Therefore, the identification of a new point, e.g., a new event along time, or a new town along a railway, could be described as

$e \rightarrow e\ i\ e$ (S1)



where

$e\ i\ e \equiv e \beta i \beta e$

Actually, the structure described by (S1) is a dual structure, since it concurrently generates two different collections of elements, i.e.,

... i, i, i ...
... e, e, e, ...

Rather counterintuitively, the rule (S1) does not describe the structure of events along time as the result of assembling additional events from the present into the future, but rather as the result of nesting new events within time spans. This is what makes it possible for a receiver to identify and store previously unidentified events in the past. It implies a construal of time not as a repeated realization of future events —as the physicist Eddington put it, "events do not happen: they are just there, and we come across them" [16]—, but as a pre-existing blank stretch into which events can be indefinitely inserted [27].

The rules (S0) and (S1) provide a means for a sender to describe a representation by means of symbols and rules in such a way that, however the sender updates it, the receiver's description will be consistent with it. Short of any geometric referents, the receiver may have no way to know what the received symbols refer to, but could nonetheless make use of a number of agreed rules to reconstruct their configuration.

The structures just described are based on intrinsic features of the geometry of things as we perceive them, and

are indeed characteristic and irreducible. You can pick one olive with a toothpick or two with a fork, but no matter what you do you will never be able to simultaneously pick more than one olive with a toothpick.

The configurations described by the rules (S0) and (S1) are irreducible, but not incompatible with each other. For example, on a map where towns have been represented as dots and the remaining surface as *U*, the items *Vienna* and *Rome* could be connected through the denotational path

k_{02}(Vienna, U) k_{20}(U, Rome)

If those towns happened to be linked by a railway R, then they could alternatively be connected as:

k_{01}(Vienna, R) k_{10}(R, Rome)

In both cases the category of towns would be the same, but the structure used to represent that category would be different.

XII. SEMANTIC CLUSTERS

Interestingly, a particular kind of category clusters can be derived by combining (S0) and (S1), resulting in expressions consistent with evolution verbs. The following example describes the concept of movement, but the same configuration could also be used to describe color change, or growing/shrinking (Figure 3).

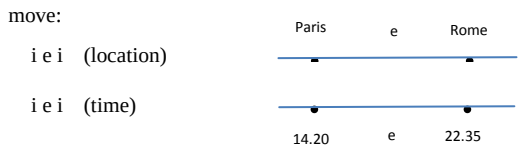


Figure 3. Category cluster describing movement

Similarly, a combination of the configuration above and the configuration described in Figure 1 would result in a semantic cluster from which a number of syntax expressions related to entering/exiting could be derived, i.e., see Figure 4.

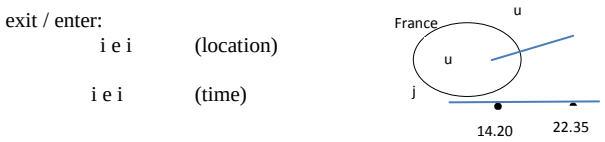


Figure 4. Category cluster describing entering-exiting

The potential for data items to be represented as components of category clusters, and therefore to generate syntax based on their semantic relations, opens the way to a number of practical applications. A few examples may illustrate this.

1. *N-tuple concepts*

In Section IV, n-tuple concepts have been described as a particular case of category clusters. They constitute a major part of NLs lexical inventory, mostly as nouns, verb roots, or, depending on the specific language, their lexical/morphological equivalents.

Two kinds of n-tuple concepts are of particular interest: (i) standalone items that can move freely, i.e., having a location that may change along time, and (ii) items that can only move with the terrain (i.e., through deformation of the surface or volume they are part of), e.g., features of a landscape. A mountain's location, for example, might change relative to other features in the landscape, but the landscape's denotational relations should not be expected to change.

Fixed location n-tuples and movable location n-tuples have intrinsically different properties, and therefore they have to be described as intrinsically different combinations of categories. If we denote them as resp. fixedQ and looseQ, then

fixedQ	q0 ∧ (q∧time) ∧ loc
looseQ	q0 ∧ (q∧time) ∧ (loc∧time)

where *loc* denotes an S0 spatial location, *q* denotes a combination of categories that may or may not evolve along time (e.g., color, or size), and *q0* denotes an instance that unequivocally characterizes the n-tuple it is part of. For material objects, *q0* is usually a shape, and generally does not have a specific label, e. g.

duck	duckY∧ (q∧time) ∧ loc
hill	hillY∧ (q∧time) ∧ loc

The labels *duckY* and *hillY* have been used to label two instances of the category *shape* that do not have a lexical correlate in English, i.e., to fill a lexical gap. In a number of languages, such instances can only be indirectly referred to, e.g., as 'the shape of a duck'.

The above description means that, in static terms, the concept 'duck' could be decomposed into a number of attributes (e.g., the categories *color*, *shape*, *size*, etc.), while each instance of a duck, i.e., each combination of those attributes, could be associated to a point in space. If those attributes are in turn combined with a space∧time configuration, i.e.,

duck	duckY∧ (q∧time) ∧ (loc∧time)

then the duck will be able not only to evolve, but also move and have a history. Concepts like *hill*, construed as fixed objects, could not move independently from their location, so they could only be described in static terms. Insofar as denotational configurations can be combined, a dynamic description of a hill would always be possible, which of course would be no news for a cartoonist.

2. *Geolocation*

Geolocation data could be represented as a category cluster by using the category structures derived from its components, e.g., see Figure 5.

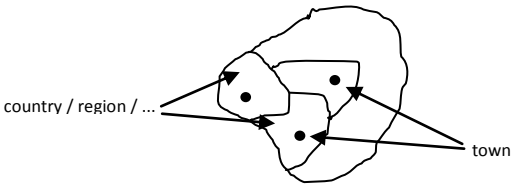


Figure 5. Geographical map categorization

For example, from the categories

Lisbon / Madrid / Rome / ...
Lisbon / Porto / Coimbra / ...
Portugal / UK / Austria / ...

a number of spatial relations could be identified. For example, if a point in a map is denoted as *x*, then the adjacency relation k_{20} could be associated to the English preposition 'in', e.g.,

in(x, Lisbon)
in(x, Portugal)
in(x, Europe)
in(Lisbon, Portugal)
in(Portugal, Europe)

The expression in(x, Lisbon) above is based on the fact that a town can be represented either as a one-dimensional point or as a two-dimensional area.

Spatial concepts potentially used by geolocation applications could be additionally defined based on quantitative-range labelling (e.g., *near*, *far*, *distance*) or on specific configurational features (e.g., *North*, *South*, *latitude*, *equator*). As has been shown, movement concepts could also be incorporated by means of the (S1) updating rules.

Similarly, time concepts could also be categorized in different ways, i.e., represented by means of different configurations, e.g., see Figures 6 and 7.

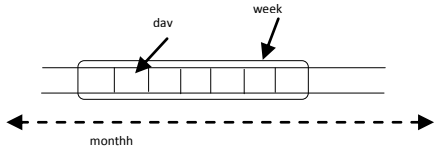


Figure 6. Time categorization (a)

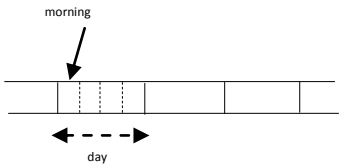


Figure 7. Time categorization (b)

Concepts such as *before*, *after*, *during*, *yesterday*, etc. would naturally emerge from such representations, and could then be used, together with spatial and/or movement representations, to derive syntax expressions with querying and/or updating purposes.

3. Meteorology

Representations describing movement could also be used to describe the evolution of meteorological variables, just by replacing spatial categories with categories derived from meteorological variables, e.g., see Figure 8.

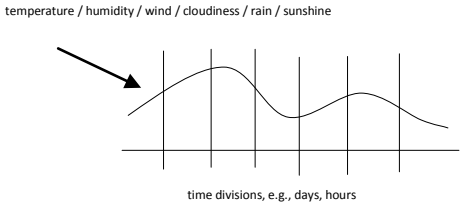


Figure 8. Evolution of meteorological variables

Expressed as category clusters, the values of the different variables would be represented as, e.g., Figure 9.

...	v_1	v_1	V	v_2	...
...	T	t_1	T	t_2	...

Figure 9. A category cluster representing the evolution of meteorological variables

where v_1, v_2 would denote individual, point-like values of the relevant variable, while *V*, *T* would generically denote resp. value ranges and time spans. Thus, in Figure 9 the value v_1 would not change during a time span *T* until the time t_1, after which it would evolve to v_2 along a range of values *V* during a time span *T*.

As has been noted, both time and value ranges could be quantified, but for the purposes of generating syntax expressions that would not necessarily be a requirement.

The above examples show that there are a number of domains where data items —whether concept attributes or measured values— can be categorized and represented as category clusters. By means of the expressions (6) and (9), such category clusters have the potential to generate as many syntax constructions as could possibly be derived from them, even if some of their elements may not have been labelled and/or conceptually identified. The filling of lexical gaps, and the development of novel theoretical frameworks, are just natural consequences of that potential.

Since the advent of smartphones, we are surrounded by data. We can wirelessly access GPS and geographical data practically anywhere anytime, as well as train/flight/bus timetables, food recipes, health tips, weather forecasts, etc. By structuring and combining those data in terms of category clusters, they could be endowed with a potential to exchange (i.e., generate and decode) information through possibly any existing NL. Furthermore, their basic components would be derived from objective and irreducible properties of space and time, and would therefore neatly fit into the definition of semantic primitives.

XIII. DATABASES AS CATEGORY CLUSTERS

A variety of approaches to communication through different arrangements of data have been described [3][11], including based on ontologies [49][5], the Semantic Web [34], and even neural networks [38]. However, neither ontologies nor databases or, for that matter, n-tuple concepts are explicitly conceived as category clusters [12]. They may contain symbols denoting categories, but in general they do not acknowledge the fine structure provided by semantic clusters, whether by identifying irreducible relations or by decomposing extant relations into irreducible ones. Furthermore, some of the categories identified are only implicitly based on spontaneous categorization, and their updating structure is not formally acknowledged and, consequently, often not used.

An interpretation of databases as category clusters, i.e., reflecting category structures and semantic relations, would make it possible to directly generate and decode a much larger and richer repertoire of string syntax —and hence, conventional syntax— expressions. It would therefore provide a more natural way to exchange information with conventional language users by using string syntax as an intermediary. This can be summarized as follows.

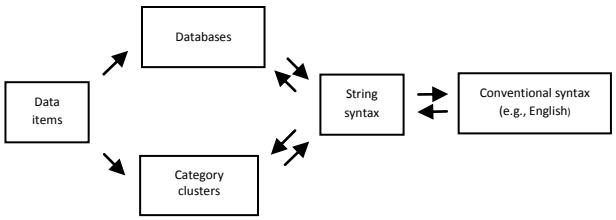


Figure 10. Two alternative data arrangements for information exchange with conventional language users

The expressions (11) and (12) provide a means to resp. update and query a communicating party through string syntax, provided that the sender's and the receiver's data arrangements are consistent with each other. Therefore, whatever the spatial configuration of a database, string syntax communication will be possible whenever the database's content can be interpreted in terms of categories and category clusters.

In a database *D*, an empty table T_θ consisting of the columns C_1, C_2, ... could be interpreted as a combination of the associated categories C_1, C_2, ..., and any instantiation of that combined category would describe a row (or potential row) in T_θ. For example, let the table T_k consist of the columns 'name', 'age', and 'address'. Because each of these can take *any* value within its respective scope, they can also be construed as categories, i.e.,

$$\text{name}() \wedge \text{age}() \wedge \text{address}() \;>>\; G_\theta$$

where G_θ would be a category cluster associated to T_θ. By specifying values for those columns, a number of rows would be obtained, e.g.,

$$\text{name(Oz)} \wedge \text{age(39)} \wedge \text{address(7th Av.)} >> \text{row}_1$$
$$\text{name(John)} \wedge \text{age(54)} \wedge \text{address(97 St.)} >> \text{row}_2$$
$$\text{name}() \wedge \text{age(33)} \wedge \text{address(221B St.)} >> \text{row}_3$$

Now, if we use the connective $\vee$ to link the rows above, then the table T_θ could be interpreted as a category ambiguously referring to any of its rows. If we use the connective $\wedge$ instead, then T_θ could be interpreted as a combination of rows, i.e.,

$$\text{row}_1 \vee \text{row}_2 \vee \text{row}_3 \vee \ldots >> \text{employee}$$
$$\text{row}_1 \wedge \text{row}_2 \wedge \text{row}_3 \wedge \ldots >> \text{employees}$$

In string syntax, both rows and cells within a row could be referred to by means of (6) and (9), e.g.,

$$\text{employee } [r_2 \text{ age(33)}]$$
$$\text{age } [r'_2 \text{ employee } [r_3 \text{ 221B St.}]]$$

where the relations r_2, r'_2, r_3 would be defined according to (4) and (5). In the general case, communication between a database and a user could be established in either direction as follows:

A. User to database

By reversing the rules used to derive conventional syntaxes, messages sent by users to a database *D* for updating or querying purposes could be expressed in string syntax by means of resp. (9) or (10). Such messages could be processed at *D* insofar as its tables could be interpreted as category clusters and those clusters would be consistent with the sender's. When that is the case, updating and querying could be interpreted in *D* as follows

String syntax	Database operation	
$G\ [r_m\ !u]$	$N_1(u_1) \wedge \ldots \wedge N_m(u \leftarrow)$	(13a)
$!G\ [r_m\ u]$	$N_1(u_1) \wedge \ldots \wedge N_m(u) \leftarrow N_m(u)$	(13b)
$?G\ [r_m\ u]$	$N_m(u) \rightarrow N_1(u_1) \wedge \ldots \wedge N_m(u)$	(13c)
$?C_m\ [r'_m\ G]$	$N_1(u_1) \wedge \ldots \wedge N_m(\rightarrow u)$	(13d)

where the symbols ← and → denote resp. the incorporation of a new item and the identification of an extant item. The updating operation would add resp. a value or a row to D, while the query would prompt D to identify resp. an item or a table, and then send the result to the querying party.

Therefore, to be able to process string syntax expressions, a database should be configured so that either (a) the column headers in its tables reflect categories potentially referred to by the user, or (b) a sub-table could be identified in D for each category cluster that might be referred to by a user.

This is not uncommon. Meteorological and geolocation databases usually record data expected to be of interest for the general user, and databases containing spatial/temporal data most often lend themselves to semantic interpretation. As an example, let us define the category cluster G as in Figure 11:

...	h_1	h_1	H	h_2	...
...	T	t_1	T	t_2	...

Figure 11. Space-time category cluster

This could be interpreted as describing a stay at the location h_1 for an indefinite time *T* until the time t_1, then some movement along some distance *H* during an indefinite time span *T*, and then the presence on a fixed location h_2 at the time t_2. From that cluster, the subclusters

h_1	H
t_1	T

H	h_2
T	t_2

could be denoted resp. as G_{depart}, G_{arrive}, *implying* the relations

from(G_{depart}, loc)
at(G_{arrive}, time)

The above relations could be used to construct a number of useful string syntax expressions, e.g.,

G_{depart} [from Rome]
G_{arrive} [at 09:23]

and therefore also updating and querying expressions, e.g.,

?time [r'$_2$ G_{arrive} [to Rome]]

For a database to be able to interpret such expressions, the adjacency relations in the sub-table

origin	destination	departure	arrival
Bonn	Rome	20:15	22:30

should be reconfigured so as to reflect the semantic relations in G, e.g.,

[origin]	H	[destination]
[departure]	T	[arrival]

so that, e.g., the sub-table

H	h_2
T	t_2

could be associated to the category cluster

G_{arrive}(loc, time)

The reconfigured table in *D* is actually a three-dimensional table, where the original columns are now arranged differently, i.e., only the topology of the table has been changed.

B. Database to user

The correspondences (13a-b) could reciprocally be used by *D* to derive reports expressed in string syntax, i.e.,

Database operation	String syntax
$N_1(u_1) \wedge ... \wedge N_m(\rightarrow u)$	G [r_m !u]
$N_m(u) \rightarrow N_1(u_1) \wedge ... \wedge N_m(u)$	!G [r_m u]

that would prompt the receiver to update its representation in response to the query previously sent, or by, e.g., a geolocation algorithm intended to keep a user updated on his surroundings.

An example would hopefully illustrate the reporting process. In a meteorological database *M*, the column headers 'temp', 'humidity', 'loc', and 'time' could be associated a combined category that a user would interpret as a number of variables describing different weather states, i.e.,

Column headers	Combined category
temp humidity loc time	temp $\wedge$ humidity $\wedge$ loc $\wedge$ time

A query intended to find out, e.g., the temperature in Paris at 22:05 would be expressed in string syntax as

?temp [r_1 Paris] [r_2 22:05]

In response to that query, the database would locate the row *R* having 'Paris' under the header 'loc' and '22:05' under the header 'time'. It would then retrieve from that row the cell under the header 'temp', and express the resulting value in string syntax as

R [r_3 !33ºC] [r_1 Paris] [r_2 22:05] (14)

If we use English words for the subindices, then we can write

r_1	r_{in}
R	$R_{a\text{-}row\text{-}in\text{-}this\text{-}database}$
r_2	r_{at}

A few translation rules, together with (7), would convert (14) into the conventional syntax expression

the temperature from a row in this database in Paris at 22:05 is 33ºC

However, the receivers need not even know that the data has been retrieved from some table in the source database. They have chosen to ask the source because they trust it to output reliable data. Therefore, the source might safely decide to just translate

the temperature in Paris at 22:05 is 33ºC

This omission might seem like a trick shrewdly devised to get the desired result. On the contrary, it is an information

compression device routinely used by natural language speakers. Consider just a few examples.

- the kitchen [of our house] is in the ground floor
- I can see the airport [of Beijing] now
- the book [you expressed an interest to buy three minutes ago] is *Finnegans Wake*

XIV. Potential Tools for Deafblind Users

One sector of the population that could potentially benefit from this framework are deafblind users. DeafBlind People (DBP) are affected by different degrees of visual/auditive impairment. They communicate through a surprisingly wide —and quite imaginative— diversity of languages and communication media [42][46], and have a severely limited access to the perceptual world [33]. Their access to information is generally limited to communication with other human beings through various, often non-standard, languages. Therefore, they generally require a human intermediary to communicate with the external world, as well as to find their way around. Attempts to facilitate DBP communication have been described, including based on gesture recognition [2][29], tactile messages [8], human-robot interaction [39], and many others. However, deafblind people's familiarity with abstract concepts is limited, which poses a formidable challenge [9].

In that respect, the potential benefits of the ideas presented in this paper are threefold, i.e., (a) string syntax could be used as a bridge between databases and DBP languages, which could give DBP users access to a wealth of external information otherwise inaccessible to them; (b) because semantic clusters can be used to generate and decode syntax expressions, the recognition of such structures, and the knowledge of the rules to deal with them, could facilitate DBP language learning and comprehension, and enhance their conversational skills; and (c) the possibility to spatially represent semantic relations could provide DBP with an invaluable tool to enhance their cognitive universe in a more systematic way than what has been achieved to date [13][48]. The first two benefits could be attained resp. through parsing and training. The third one will be discussed below.

The structure of perceptions plays a key role in the formation of concepts. As an example, people who have experienced long-term visual deprivation and are given access to retinal perceptions initially fail to see those perceptions in a structured way [20]. Nonetheless, insofar as perceptions can be represented in terms of spatial adjacency relations, they might be accessible anyway. No blind person can perceive colors, but a structural description of a rainbow or, for that matter, of the color spectrum could be apprehended by them and labelled in a way consistent with non-visually impaired people's. Such possibilities open the door to a number of tools aimed at a cognitive enhancement of DBP. While DBP lack the input to construct some sensory representations, such representations could possibly be taught to them, thereby helping them not only to find their way around without human assistance, but also to enhance their knowledge about the world they live in.

Actually, language learning and cognitive enhancement would go hand in hand. Both would rely on three components dealing with the aspects of resp. cognition, output, and input, namely: (a) recognizing the structure of basic categories, as well as a number of elementary category clusters; (b) learning to associate lexical tokens to semantic representations or parts thereof; (c) recognizing the basic components of string syntax, i.e., categories, instances, and relations, as well as the spontaneous categorization mechanism.

The realization that a number of categories can be updated and assembled, and of how it can be done, should in turn lead to the realization that semantic gaps can be filled, and that more complex concepts can be envisioned, whether they have a material correlate or are a personal creation. This should make DBP aware that the both the material and mental worlds are ever larger and, with the right tools at hand, it can be explored.

A roadmap along those lines could only be sketched within the scope of this paper. In the examples below, the teaching methods suggested appear in italics, while the teacher's prompts are shown between asterisks, and lexical tokens are enclosed in angle brackets. The symbol :: denotes the expected association between the prompts and the corresponding lexical tokens.

Category **animal** (e.g., toys)
Tactile recognition
<animal><x>
where <x> :: *duck*/*bird*/*turtle*/...

Category **shape**
Tactile recognition
<shape><x>
where <x> :: *duck*/*square*/*circle*/...

Category **size**
Tactile recognition
<large><x>/<small><x>
where <x> :: *duck*/*square*/*circle*/...

Category **up**/**down**
Hand position
<x><up>/<x><down>
where <x> :: *duck*/*box*/*hand*/...

Spatial updating
Tactile recognition
<y><next to><x>
where <x>,<y> :: *me*,*table*/*table,*you*/...

Time updating
Tactile exploration (e.g., on a clock face)
<y><then><x>
where <x>,<y> :: *<sleep>*,*<wake up>*/...

Category **still**/**moving**

Tactile recognition
<x><still>/<x><moving>
where <x> :: *<rabbit>*,*<car>*/...

Having identified space and time concepts and their respective updating structure, a possible course of action could be for the DBP learner to be presented the semantic cluster in Figure 11. From that cluster, a number of subclusters representing concepts, e.g., *move*, *stay*, *before*, *from*, *at*, etc. could be identified, and their lexical correlates could be used to assemble a number of syntax string expressions, as described in Section XIII.

Regarding the possible implementations, a portable device with a haptic interface, which could physically change hands to send and receive messages by other human parties, would arguably provide a higher degree of autonomy than garments or other wearable devices. Furthermore, the interface provided by a portable device would facilitate a more sophisticated interaction, particularly for the purposes of language exchange, and language and concept learning. It could also be a means, or at least provide a stimulus, for the users to replace their sign/haptic language or dialect with string syntax as a user-friendly, universal language. Its three basic elements, i.e., categories, instances and relations, could be readily expressed by means of haptic icons, and its syntax rules are simplest and intuitive.

Human language is a vast field, encompassing all kinds of conceivable concepts. Therefore, any communication project could only be realistic if constrained to a specific, clearly delimited concept domain. In view of this, a possible plan aimed to learning and communication should initially consist of a number of basic semantic fields plus a roadmap to expand such fields to other semantic domains. Actually, a significant problem might be the identification of clear boundaries, considering the all-pervading nature of semantic notions.

A number of tools to be initially developed to implement such an interface would be as follows:

COST: A parser to translate [simple] conventional syntax (CO) expressions into string syntax (ST) expressions
STCO: A translator from string syntax (ST) to conventional syntaxes (CO)
LESE: A dictionary associating lexical tokens (LE) to semantic configurations (SE)
META: A dictionary of metalanguage signs to instruct the recognition of semantic tokens and their association to string syntax components
DASE: A module to rearrange database data (DA) into identifiable semantic configurations (SE)
LEX: A number of limited lexicons from different domains. Initially, they might include meteorology (MET), and geolocation (GEO). As a potential further addition, static and dynamic in-out concepts (IOC) could also be explored

The author has already designed an interface along those lines. However, a detailed description of that interface would be beyond the scope of this paper.

CONCLUSION

A representational approach to data, together with the categorial structure implicit in natural languages, can be seen as the components of a formal framework that integrates syntax and semantics under a single theoretical construct. Within that novel syntax/semantics integrated framework, human communication could be achieved by structurally representing either internal thoughts or external perceptions/data as category clusters. In such a representation, the category-instance relation could be used to locally refer to individual components through a disambiguation process, which could be expressed in a particular string form.

The syntax informing such strings has been shown to be consistent with conventional languages, as well as databases and various representational implementations. Furthermore, the identification of formal structures underlying categories lends category clusters an objective semantic quality, and endows them with the potential to generate syntax expressions.

This unification of syntax and semantics into one single model could be the basis for an interface to be designed, among other purposes, to: (a) operate as a universal translator; (b) derive language expressions from spatial representations, and conversely, extract representations from syntax strings; (c) rearrange databases in a representational format; and (d) give sensory impaired people a more extensive access to the external world.

Future work along those lines would include the implementation of a number of interactive database-user interfaces, e.g., for geolocation purposes, flight/train data querying at resp. airports/train stations, etc., with the aim of progressively enlarging their scope and incorporating ever more complex data sources. The optimization of such interfaces would also be interactive, and essentially not different from the dispelling of misunderstandings in colloquial language.

The development of an interactive methodology along the lines sketched in Section XIV would also be a promising tool to enhance the cognitive universe of deafblind people.

ACKNOWLEDGEMENTS

Some notions presented in this paper, and specifically the formalization of the concept of denotational jigsaw, might not have materialized without the critical comments of Ernesto Sánchez de Cos, who forced me to delve deeper into ideas that initially were rather vaguely expressed. The patience and kindness of Sándor Darányi, always willing to pay attention to my ramblings, rain or shine, were also invaluable to me, as well as his critical comments and his emotional support during the long and painstaking mental process that led to the completion of this paper. My sincere thanks to both.

REFERENCES

[1] C. Seror, "A Data Referencing Formalism for Information Exchange between Deafblind People and Databases," SEMAPRO 2019, The Thirteenth International Conference on Advances in Semantic Processing, 2019.

[2] G. Airò Farulla et al., “Real-Time Single Camera Hand Gesture Recognition System for Remote Deaf-Blind Communication,” International Conference on Augmented and Virtual Reality AVR 2014: Augmented and Virtual Reality, pp. 35-52, 2014.

[3] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch, “Natural language interfaces to databases - An introduction,” Natural Language Engineering, 1(1), pp. 29–81, 1995.

[4] L. W. Barsalou, “Perceptual symbol systems,” Behavioral and Brain Sciences 22, 577–660, Cambridge University Press, 1999.

[5] J. A. Bateman et al. “A linguistic ontology of space for natural language processing,” Artificial Intelligence, 174.14: 1027-1071, 2010.

[6] R. J. Bobrow and R. M. Weischedel, “Challenges in Natural Language Processing,” Cambridge University Press, 1993.

[7] M. H. Bornstein and C. Mash, “Experience-Based and On-Line Categorization of Objects in Early Infancy,” Child Development, Vol. 81, Issue 3, pp. 884-897, 2010.

[8] S. Brewster, and L. Brown, “Tactons: Structured tactile messages for non-visual information display,” in 5th Conf. on Australasian User Interface, ACM, pp. 15-23, 2004.

[9] S. M. Bruce, “The Impact of Congenital Deafblindness on the Struggle to Symbolism,” International Journal of Disability, Development and Education, 52:3, 233-251, 2005.

[10] A. Cahen and M. C. Tacca, "Linking perception and cognition," Frontiers in Psychology, 2013.

[11] N. Caporusso et al., “Enabling touch-based communication in wearable devices for people with sensory and multisensory impairments,” in: 1st International Conference on Human Factors and Wearable Technologies, pp. 149-159, 2017.

[12] E. F. Codd, “A Relational Model of Data for Large Shared Data Banks,” IBM Research Laboratory, San Jose, California, 1970.

[13] W. S. Curtis, E. T. Donlon and D. Tweedie, “Learning behavior of deaf-blind children. Education of the Visually Handicapped,” 7(2), American Psychological Association, pp. 40-48, 1975.

[14] M. E. D'Imperio, “Data structures and their representation in storage,” Annual Review in Automatic Programming, Volume 5, Part 1, 1969, pp. 1-75, ISSN 0066-4138.

[15] J. Davidoff, “Language and perceptual categorisation,” Trends in Cognitive Sciences, Volume 5, Issue 9, 1 September 2001, pp. 382-387.

[16] A. S. Eddington, “Space, Time and Gravitation. An outline of the general relativity theory,” Cambridge University Press, 1920.

[17] N. Evans, “Dying Words: Endangered Languages and What They Have to Tell Us,” Wiley-Blackwell, 2009.

[18] D. L. Everett, “Cultural Constraints on Grammar and Cognition in Piraha,” Current Anthropology, Volume 46, Number 4, 2005.

[19] G. Fauconnier, E. Sweetser (eds.), “Spaces, Worlds, and Grammar,” The University of Chicago Press, pp. 1-28, 1996.

[20] I. Fine et al., “Long-term deprivation affects visual perception and cortex,” Nature Neuroscience, Volume 6, No. 9, 2003.

[21] G. Frege, “Über Sinn und Bedeutung,” in: Zeitschrift für Philosophie und philosophische Kritik. Vol. 100, pp. 25–50, 1892.

[22] C. Freksa, “Spatial and temporal structures in cognitive processes,” Lecture Notes in Computer Science, Vol. 1337. Springer, 1997.

[23] A. Gatt and E. Krahmer, “Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation,” Journal of Artificial Intelligence Research 61, 65-170, 2018.

[24] G. W. Gifford III, K. A. MacLean, M. D. Hauser and Y. E. Cohen, “The Neurophysiology of Functionally Meaningful Categories: Macaque Ventrolateral Prefrontal Cortex Plays a Critical Role in Spontaneous Categorization of Species-Specific Vocalizations ,” Journal of Cognitive Neuroscience, Volume 17, Issue 9, pp. 1471-1482, 2005.

[25] N. Giret, F. Péron, L. Nagle, M. Kreutzer, D. Bovet, “Spontaneous categorization of vocal imitations in African grey parrots (Psittacus erithacus),” Behavioural Processes, Volume 82, Issue 3, pp. 244-248, 2009.

[26] A. Gopnik and A. Meltzoff, “The Development of Categorization in the Second Year and Its Relation to Other Cognitive and Linguistic Developments,” Child Development, Vol. 58, No. 6, pp. 1523-1531, 1987.

[27] U. Hasson, E. Yang, I. Vallines, D. J. Heeger and N. Rubin, “A Hierarchy of Temporal Receptive Windows in Human Cortex,” Journal of Neuroscience, 28 (10) 2539-2550, 2008.

[28] E. Horowitz and S. Sahni, “Fundamentals of Data Structures,” Computer Science Press, 1983.

[29] J. Kramer and L. Leifer, “The talking glove,” SIGCAPH Comput. Phys. Handicap 39, pp. 12–16, 1988.

[30] R. K. Logan, “What Is Information?: Why Is It Relativistic and What Is Its Relationship to Materiality, Meaning and Organization,” Information 2012, 3(1), 68-91, 2012.

[31] R. M. Losee, “A Discipline Independent Definition of Information,” Journal of the American Society for Information Science, 48 (3) 1997, pp. 254-269, 1997.

[32] D. Mareschal and P. C. Quinn, “Categorization in infancy,” TRENDS in Cognitive Sciences Vol.5 No. 10, pp. 443-450, 2001.

[33] B. Miles, “Overview on Deaf-Blindness,” Helen Keller National Center, Revised October 2008.

[34] B. Munat, “The lowercase semantic web: using semantics on the existing world wide web,” Evergreen State College, Olympia, 2004.

[35] D. G. Nash, “Topics in Warlpiri Grammar,” Massachusetts Institute of Technology, 1980.

[36] S. E. Petersen and O. Sporns, “Brain Networks and Cognitive Architectures,” Neuron, 88(1): 207–219, 2015.

[37] M. Petruck, “Frame semantics,” Handbook of pragmatics, pp. 1-13, John Benjamins Publishing Company, 1996.

[38] P. Z. Revesz and R-R. K. Veera, “A Sign-To-Speech Translation System Using Matcher Neural Networks,” https://cse.unl.edu/~revesz/papers/ANNIE93.pdf 2020.05.26.

[39] L. O. Russo et al., “PARLOMA – A Novel Human-Robot Interaction System for Deaf-Blind Remote Communication,” https://journals.sagepub.com/doi/10.5772/60416 2020.05.26.

[40] C. Seror, “Human language and the information process,” https://zenodo.org/record/3263753#.XRmuceszaUk 2020.05.26.

[41] C. E. Shannon, “A Mathematical Theory of Communication,” Bell System Technical Journal, Vol. 27, pp. 379–423, 1948.

[42] W. C. Stokoe, Jr., “Sign Language Structure: An Outline of the Visual Communication Systems of the American Deaf,” The Journal of Deaf Studies and Deaf Education, Vol. 10, Issue 1, pp. 3–37, 2005.

[43] A. M. Tozzer, “A Maya Grammar, with bibliography and appraisement of the works noted,” Papers of the Peabody Museum of American Archaeology and Ethnology, Vol. IX, p. 55, 1921.

[44] R. Vigo, “Representational information: a new general notion and measure of information,” Information Sciences 181, 4847–4859, 2011.

[45] R. Vigo, “The GIST of concepts,” Cognition, Vol. 129, Issue 1, Elsevier, pp. 138-162, 2013.

[46] YouTube, "Sistemas de Comunicación para Personas Sordociegas," https://www.youtube.com/watch?v=SpcsbgM1TIc 2020.05.26

[47] R. Wille, "Formal Concept Analysis as Mathematical Theory of Concepts and Concept Hierarchies," Formal Concept Analysis: Foundations and Applications. Ganter, B., Stumme, G., Wille, R. Eds. Springer, pp. 1-33, 2005.

[48] K. Wolff Heller, P.A. Alberto, and J. Bowdin, "Interactions of communication partners and students who are deaf-blind: A model," Journal of Visual Impairment & Blindness, 89(4), pp. 391-401, 1995.

[49] M. Zerkouk, A. Mhamed, and B. Messabih, "A user profile based access control model and architecture," http://www.airccse.org/journal/cnc/0113cnc12.pdf 2020.05.26.

[50] F. Mesquita Carvalho and N. Kriegeskorte, "The Arrow of Time: How Does the Brain Represent Natural Temporal Structure?," Journal of Neuroscience, 28 (32) 7933-7935, 2008.

Improve Decision Support System Operations of Real-Time Image Classification Utilizing Machine Learning and Knowledge Evolution

David Prairie
Electrical and Computer Engineering Dept.
University of Massachusetts Dartmouth
Dartmouth, Massachusetts United States
Email: dprairie@umassd.edu

Paul Fortier
Electrical and Computer Engineering Dept.
University of Massachusetts Dartmouth
Dartmouth, Massachusetts United States
Email: pfortier@umassd.edu

***Abstract*—This paper delves into the generation and use of image classification models in a real-time environment utilizing machine learning. The ImageAI framework was used to generate a list of models from a set of training images and also for classifying new images using the generated models. Through this paper, previous research projects and industry programs are analyzed for design and operation. The basic implementation results in models that classify new images correctly the majority of the time with a high level of confidence. However, almost a quarter of the time the models classify images incorrectly. This paper attempts to improve the classification accuracy and improve the operational efficiency of the overall system as well.**

Keywords-component; Machine Learning; Tensor Flow; Image Classification.

I. INTRODUCTION

Presented in this research paper is a new and novel approach to machine learning design that maximizes the balance between accuracy, efficiency, solution justification, and rule evolution. This paper is a more complete and informative description of the research presented at the IARIA Data Analytics conference [1]. This research improves four factors of machine learning within a single design. During this research, traditional open source machine learning methodologies were altered to improve different aspects of machine learning, tested against a single application. These traditional methodologies were integrated using ensemble methods for enhancing the performance along with providing justifications for the classification results. This paper discusses the results, data used, the analysis, and validation methodologies used.

This research attempted to find an optimal balance between maximizing a system's accuracy and minimizing a system's time and space requirements. As shown in Figure 1, these three attributes are directly correlated with each other and the system integrator needs to determine the trade-off required for the implemented system.

During this research the Identifiable Professionals (IdenProf) Dataset was used for training and validating models. The dataset contains ten image sets of distinguishable professions, each set containing 900 images for training and 200 images for validation per profession. The expected outcome of this research is a two part system capable of analyzing a real-time feed and perform profession classification based on a remotely generated model.

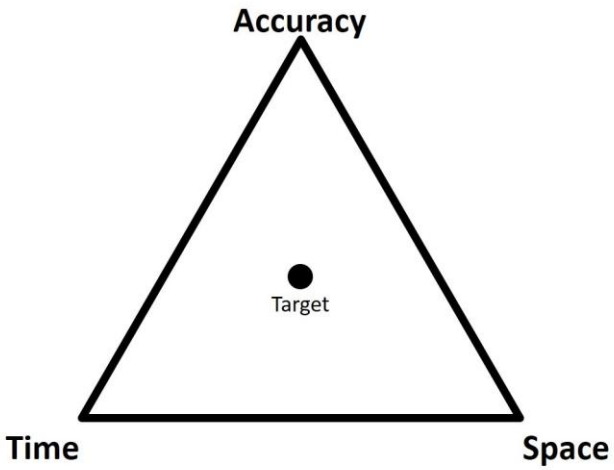


Figure 1: System Efficiency Tradeoffs

This research aimed at answering the following four questions based upon improved decision support system operations. Investigations and implementations conducted comprised of different components of the Decision Support System (DSS). This research reviewed prior research in the machine learning technical field and built upon previous research to answer the following hypothesis:

1. Utilizing a knowledge base, can a DSS be implemented to minimize the time and space requirements, while maximizing the accuracy of the suggested solutions?
2. How can a knowledge base be implemented to improve the overall accuracy of the system?
3. Is a DSS able to provide solution justification to the user, enabling users to have trust in the answers being provided?

4. Is it possible to improve rule evolution without needing to store locally all previous cases for doing rule re-validations?

This paper is broken up into a Background section, where high-level aspects of machine learning and knowledge-bases are discussed. The Methodology is discussed following the Background section. Following the Methodology section the preliminary results are discussed in the Data Analytics and Results sections. Finally, the conclusion discusses planned future work and recommended further work.

II. BACKGROUND

Knowledge base systems enable problems to be quickly and accurately solved based on previous cases. Knowledge base systems also allow for problem solving of very complex situations at speeds humans alone would not be able to achieve at such accuracies. Throughout the last 20 years, knowledge bases have grown in applications to assist in everyday tasks. More recently, IBM's Watson, an Artificial intelligence supercomputer, is in the limelight through its uses in the medical arena and in personal taxes with H&R Block. Other applications of machine learning have been completed or are being developed include detecting insider threats, big data analytics, market analysis for proposals, condition-based maintenance, and diagnostics in the medical field.

In the medical industry, Watson has proven capable of making the same recommended treatment plans as doctors 99% of the time. Unlike traditional human doctors, Watson can use all available medical resources when making a patient's diagnosis. By having such vast amounts of knowledge, Watson can provide treatment options doctors may miss. Watson utilizes powerful algorithms and immense computing resources to analyze all medical relevant data to find "…treatment options human doctors missed in 30 percent of the cases" [3]. Since Watson has so much computing power it is able to determine treatment plans for patients faster than human doctors could, allowing doctors to put patients on treatments faster with the intervention of Watson. Watson is one example of how knowledge base systems positively benefit society by efficient and accurate problem solving of complex problems. Additional research into knowledge bases will allow them to problem solve faster, with more accuracy, and with less compute power requirements.

Condition-Based Maintenance, shown in Figure 2, is the method of monitoring a system's components to determine what level of maintenance is required for the system to remain functioning. Using a Condition-Based Maintenance system for managing maintenance events allows professionals to be proactive with performing maintenance activities versus being reactive. Reactive maintenance involves replacing components after they already failed, causing system downtime. When a system goes down for unscheduled maintenance or repairs, many different repercussions can occur depending on the affected system's role. Using air traffic control centers as an example, any unplanned downtime has the potential to disrupt hundreds of flights and cost a significant amount of money due to flight delays [12]. Machine learning can be used to assist professionals to determine optimal maintenance schedules while minimizing system down time.

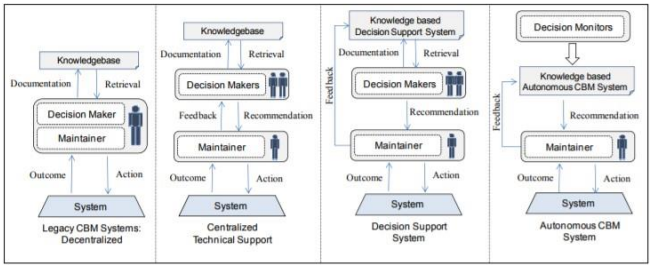


Figure 2: Condition Based Maintenance [2]

Although some of the described applications may not be as drastic as life or death medical decisions, all still can greatly affect society. Utilizing machine learning allows organizations to detect threats, conduct predictive maintenance, and perform many repeatable decision-making tasks consistently and efficiently. By allowing a machine to learn over time through historical cases and building a knowledge base, the machine allows operators to make informed decisions by providing every available piece of information. Systems are able to make decisions in a fraction of the time compared to a human expert attempting to come to the same decision, however additional advancement is needed to make machine learning more accurate and efficient. Areas needing additional inquiries include indexing algorithms, storage solutions, and finally the decision-making algorithms themselves. Machine learning is important because of the wide range of applications and benefits provided through the decision making and predictions capable. As the field advances, machines will create predictions and perform decision making faster and more completely.

A. *Watson*

Watson originally became well-known for competing in Jeopardy. Watson is a knowledge base altered for various applications including Jeopardy, medical field, and taxes. Breaking down questions from a complex human language was required for Watson to compete at the Jeopardy game [3]. The analysis of the Jeopardy questions and identifying the correct answer needed to happen almost instantaneously to compete on a high caliber level.

With the Jeopardy Challenge, Watson needed to break down questions out of human language to a format Watson could understand. The questions needed to break down into the main statement and then separate supporting statements out. "…decompose the question into these two parts and ask for answers to each one, we may find that the answer common to both questions is the answer to the original clue" [3].

In recent years, IBM altered Watson to handle taxes by collaborating with H&R Block. Although there is not much technical information available discussing the design of

Watson's work with taxes, there are a few assumptions that can be made. We would assume Watson uses a rule and case-based design. The rules would take in data on a new client, which determine what tax actions could take place. Watson would compare the new client to all previous clients allowing for more accurate and consistent tax evaluation.

B. *Deep Learning Frameworks – Tensor Flow*

Tensor Flow is a deep learning framework built on the first generation framework called DistBelief. Both frameworks were developed by Google to advance technology for the public and for use in Google's wide range of data products [4]. One of TensorFlow's major improvements over DistBelief is its ability to scale up onto large distributed hardware platforms utilizing multiple CPUs and GPUs. Tensor Flow utilizes a master orchestrator to distribute work across the number of hardware platforms available, each individual platform then breaks the work down to be solved across each system's available CPUs and GPUs.

Benchmarks conducted by Google researchers showed the Tensor Flow framework performs, as well as other popular training libraries. However, Tensor Flow did not have the best performance statistics as other libraries in the study when tested on a single machine platform [9]. Researchers at Google are continuing development in the Tensor Flow framework to incorporate additional optimization and automation to improve the performance of the framework.

C. *Rule Evolution IB1 & IB2 Algorithms*

The IB1 and IB2 algorithms are used to evolve a system's rules used for classification by incorporating new cases. The addition of more instances over time causes the machine to alter its rules to improve the probability of giving a correct prediction on future instances. Instances can either enforce existing rules or go against existing rules. Over the course of a training period, the IB1 algorithm will converge to the actual results based on altering its rules. IB1 requires data to have specific attributes, making cases distinct enough for the algorithm to learn over time. If the data does not have distinct attributes then the machine will not learn, since no strong points of comparison are available between cases [5].

A downside of the IB1 algorithm is the need to store all correct and incorrect classifications over the lifetime of the machine. The IB2 algorithm is a branch of the IB1 algorithm that does not require the storage of all classifications, only the incorrect classifications. The tradeoff of saving storage space is the increase in time required for the IB2 algorithm to learn to predict with strong accuracy [5].

During the evaluation of both the IB1 and IB2 algorithms, researchers determined both algorithms are able to achieve acceptable prediction accuracies in some situations. However, IB1 attains greater accuracies on each scenario when compared to the IB2 algorithm. The increase in accuracy for IB1 could be attributed to the storing of all classification events versus only the incorrect classifications.

D. *Ensemble Method*

Ensemble methods is the practice of implementing multiple machine learning algorithms and incorporating an additional algorithm to vote the responses to a single response. "Ensemble methods are learning algorithms that construct a set of classifiers and then classify new data points by taking a (weighted) vote of their predictions" [6]. Ensemble methods help to remove model bias and overconfidence for models against specific applications. "Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking)" [7].

Figure 3: Weather Model Example [8]

Ensemble methods are used in other industries; for example meteorologist compare numerous models to generate a cone of certainty for hurricane predictions. Figure 3 shows an example of a cone of certainty track for hurricane Dorian in 2019. This cone was generated by averaging multiple hurricane track models to a single cone. The cone shows decreasing assurance in the accuracy the further in time of the prediction.

One ensemble method, the Bayesian method, is a common practice of integrating multiple classifiers into a single classifier. The Bayesian method utilizes a weighted average method for determining the proper response. However, researchers from the University of Washington found the Bayesian method has a higher rate of error than other methods of ensemble [9].

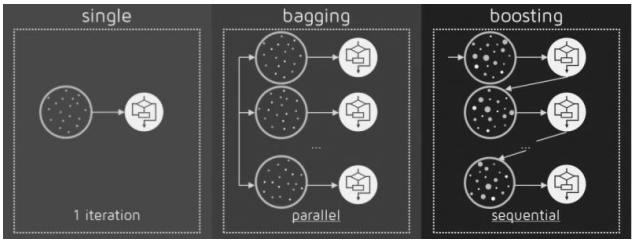


Figure 4: Ensemble Voting Methods [10]

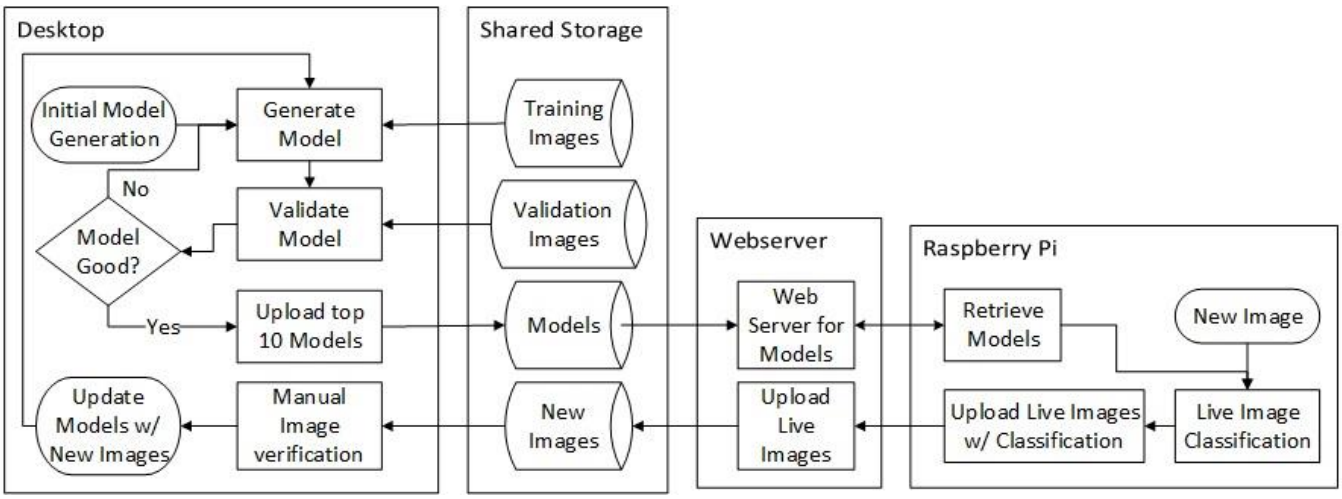


Figure 5: System Flow Diagram

Another method used when incorporating ensemble with machine learning is the bagging method. Bagging involves multiple training models on different subsets of data and integrating the outputs from each model to produce a single result [7]. Boosting is another method for using numerous predictions to create a single result. Both the Bayesian and bagging methods are depicted in Figure 4.

III. METHODOLOGY/THEORY

This section discusses the methodology used in the completion of this research. The research process followed the system engineering v-diagram during development, as shown in Figure 6.

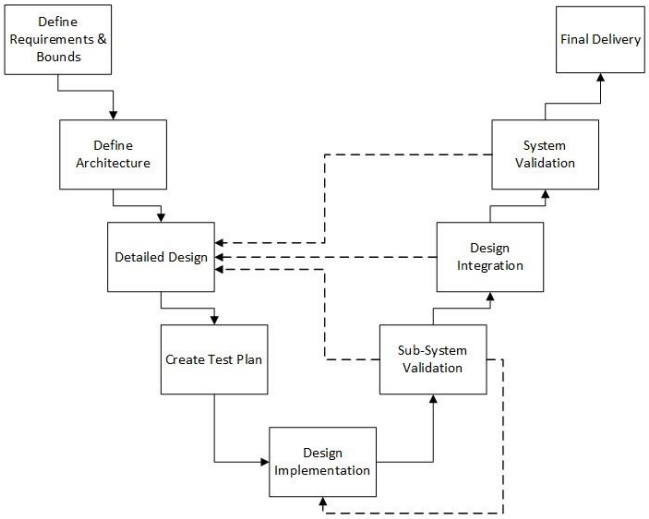


Figure 6: Design Methodology

This section is further broken into a system architecture and software architecture sections.

A. *System Architecture*

The implementation, which is shown in Figure 5, is broken into four sections; a workstation computer, web server, raspberry pi, and a shared storage box. The workstation computer contains an NVidia GTX 980ti and is used for generating models based on the training images. Once the models are generated they are stored on a centralized shared storage array.

The model generator portion of the system handles ingesting a multitude of images and generating 200 models per generation algorithm. The model generation is a compute-intensive operation, taking about 90 seconds per model at 200 models per algorithm to run, and requires a high level of resources to provide accurate results. Additional hardware options were investigated, including Amazon's Web Service and Digital Ocean's droplets. Both of these alternatives allow users to utilize a pay by use virtual Linux environment having a wide range of hardware scaling options. These options were not chosen to eliminate variables introduced by relying on another company's infrastructure.

A Raspberry Pi 3 B+ [11] was used for real-time image classification utilizing an onboard camera. During the initial testing and validation, the model generator was used. The model generator was capable of classifying a large number of images in rapid succession to validate the improvements implemented. The Raspberry Pi was best suited for completing single image classifications.

During initial testing and validation, the Model Generator was also used in place of the Raspberry Pi. Connected to the workstation is a networked HDD used for storing the generated models.

The web server is the middle point between the workstation and the raspberry pi, by serving the models generated for the Pi to download. To enable future learning from real imaging, the Pi will upload classified images to the web server for the workstation to use in future model generation.

B. Software Architecture

The following software packages and frameworks are used to generate the models for real-time image classifications and for classifying new images:

- Python 3.6
- Tensorflow-GPU
- Numpy
- SciPy
- OpenCV
- Pillow
- Matplotlib
- H5py
- Keras
- ImageAI [12]

ImageAI is an API that can generate models based on an image set and perform image classifications based on the generated models. The API can generate models using the Desenet, Inveption v3, Resnet, and Squeezenet algorithms.

The workstation utilizes the above listed software when generating the initial models used for classification. ImageAI is a wrapper framework for the rest of the libraries, simplifying the development process. The same ImageAI framework is used on the raspberry pi for real-time image classification utilizing an add-on camera board. The raspberry pi is capable of handling the classification algorithms because the model generation and model evolution is offloaded to the workstation [13]. This heavily reduces the compute requirements, enabling the mobile real-time classification.

The web server is a repository for the raspberry pi to retrieve the latest generated models and to upload classified images for further analysis. Real-time images are uploaded to the webserver for manual verification of the image's classification and are then loaded into the workstation as additional training images to evolve the models. The combination of the workstation and Raspberry Pi enables an overall system supporting model evolution while increasing the efficiency of the real-time classifier [13].

A future implementation adds a system capability of justifying the classifications provided. The current design behind the justification uses the built in confidence levels provided when classifying images. An alternative approach includes providing sample images that were classified using the same models and produced the same results.

C. Model Training

The generation of the classification models requires one data input and five configuration settings to generate the models. The script takes a folder path to a subset of the image dataset used for training as a script input. The remainder of the images is used for validating the generated models. The folder structure, as seen in Figure 7, consists of one folder per profession with each folder containing at least 200 images.

The next section of the script generates four different types of models based on four different generation algorithms. Each generation takes five configuration settings. The first one defines the number of image types, in this scenario being the ten different professions. The next input is the number of experiments, which determines the number of models to generate. The enhance_data input is an optional input, “This is used to state if we want the network to produce modified copies of the training images for better performance" [14]. The batch size input is dependent on the computing hardware available; the number is set to the maximum amount allowed by the equipment available. Finally, the show_network_summary input is used for providing detailed information to the console during model generation. The script cycles through the algorithms for generation and then generates 200 models and one JSON file per algorithm. The script serially generates the models for all four algorithms: DenseNet, Inceptionv3, ResNet, and SqueezeNet.

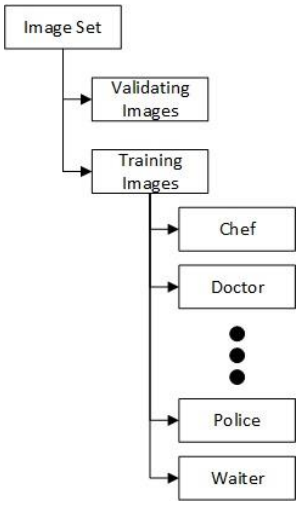


Figure 7: Image Folder Structure

Models are generated by breaking apart an image into simpler parts. These parts determine a rule set that goes into different layers. A culmination of each of these layers allows the model to make predictions based on an inputted image. The algorithm structures the layers in an optimal fashion to maximize the efficiency of image predictions.

During model generation the number of models to generate determines the number of variations the underlying algorithms with attempt. For instance, this research used 200 variations for the parameter settings producing 200 different models. The algorithms then determines an accuracy rate for each model generated, allowing the user to select the model with the highest evaluated accuracy. The generated models are given a specific naming structure for easy identification. The first parameter is an identification number ranging from one to the number of models generated. The second parameter gives the evaluated accuracy of the model, given in decimal format.

Once all the models are generated, a PowerShell script identifies the top three accurate models per algorithm. The script copies the chosen models to a separate folder for use in image classifications. When the models are generated, the calculated accuracy is added to the filename, which is how

the PowerShell script selects the top three. At this point, the models are either used for individual image classification or bulk image classification.

D. Image Classification

Classifying a single image is done through the individual image classification script. This script takes in two directory paths and a probability threshold as inputs. The directory paths contain the location of the image to classify and the models to use during classification. The probability threshold determines what predictions from the models are used in the voting. When the ImageAI algorithm attempts to classify an image with a model, the algorithm returns a single prediction with the probability of correctness. The probability threshold variable only allows predictions with greater than 80% confidence to be included in voting for the final predicted profession.

After all twelve models preform their prediction, the classifications that do not need the threshold requirements are thrown out. The remaining predictions are used in a simple majority voting scheme. The prediction with the majority of the votes from the various models is presented to the user with the average prediction confidence level and the number of models agreeing with the prediction. The script is capable of providing multiple predictions per image; however, for this application, only single predictions are needed.

E. Learning Algorithms

The ImageAI algorithm is used for the generation of the models and acts as a wrapper library to various machine learning algorithms, including Tensorflow. “ImageAI is an easy to use Computer Vision Python library that empowers developers to easily integrate state-of-the-art Artificial Intelligence features into their new and existing applications and systems" [15]. By implementing the system utilizing ImageAI, the overall development cycle was simplified by masking the low-level coding. For this system, custom recognition is utilized. However, the algorithm is capable of also performing objection recognition and live video detection [15].

F. Ensemble Methods

Two ensemble methods were used for combining the results of the individual models. The initial method used a simple majority voting scheme were each model has a single vote to the prediction. During analysis the simple majority method was altered to take into account each vote's confidence level. This weighted voting method calculated, which prediction has the highest confidence with a high number of votes. For example, if four models prediction waiter at a 95% confidence but five models predicted chef at 80% then the simple majority would produce chef as the answer, while the weighted voting would produce waiter.

IV. Data Analytics

During this research, the Identifiable Professionals (IdenProf) dataset [16] is used for evaluating the proposed changes to the ImageAI algorithm. IdenProf contains 10 distinguishable professions, listed in Table I. A sample image for each profession is shown in Figure 8. The dataset consists of over 900 images per profession used for training the system's models and an additional 200 images per profession for validating the models. All images are sized to a common pixel dimensions of 224 by 224 for uniformity. The image set has a makeup of mostly white males from the top 15 most populated countries [16], compared to other genders or nationalities. During the duration of this research project additional images can be gathered by pulling images from Google's search engine.

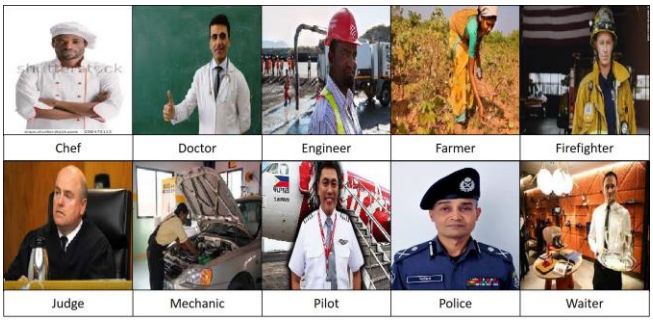


Figure 8: Sample Profession Images [16]

The experiments included testing the base algorithms against the training and validation images. These 200 images allowed analysis and validation of the models generated at all three stages of development. Additional experiments utilized the raspberry pi to simulate processing images on a low-powered machine. The models used in classifications are selected based on the assigned accuracy defined during model generation. For these experiments the models selected have over eighty percent accuracy.

Table I: Training Images Classification

Training Images Classifications	
Professions	Accuracy
Chef	74.5%
Doctor	76.5%
Engineer	86.0%
Farmer	89.5%
Firefighter	90.5%
Judge	92.0%
Mechanic	84.5%
Pilot	87.5%
Police	87.5%
Waiter	72.0%

Figure 10 depicts a collection of the test images for a pilot, one of the professions used in this research project. When running a classification against a pilot image, the system provides three results. Each result comes with a probability that the answer is correct. Typically the models generate one answer with a probability of over 95% and then the remaining two answers will make up the remaining percentage. During single image predictions, the system

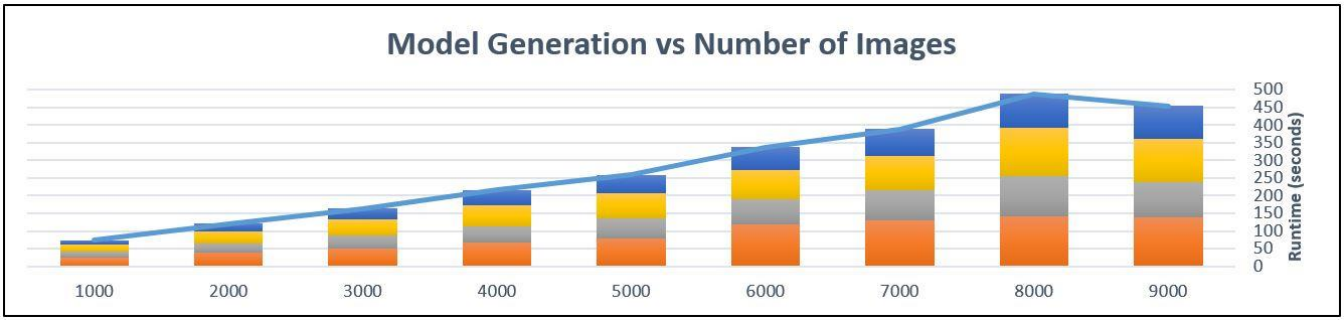


Figure 9: Model Generation Runtime

provided a profession prediction, with a confidence level, and the number of models agreeing with the answer. The confidence level is the average of the models in agreeance on the vote disclosing the models that do not meet the threshold. During this sample run three of the twelve models did not meet the required threshold and were dropped from the calculations. The remaining nine models resulted in pilot as the answer with a combined confidence of 99.99%. Similar results were found on additional tests with different images. Single image predictions was tested on both the workstation and the Raspberry Pi B+ to act as a low powered system.

Figure 10: Sample Profession Images - Pilot [16]

While testing the performance of the model generation algorithm, the runtimes for each model algorithm were compared at different image dataset sizes. Figure 9 shows the runtime of the four model algorithms and the total runtime at nine image set sizes. With the exception of the final image set of 9000, each runtime gradually increased compared to the next smallest image set.

V. RESULTS

The results were gathered through two different methods, the first conducting single image predictions and the other doing an automated bulk analysis.

A. Single Image Classification

During testing using the Raspberry Pi 3 B+ for image prediction, the Raspberry Pi required slightly different software. The Raspberry Pi required older versions of some libraries because the newer versions were not yet compatible. Figure 11 shows the results when testing the Raspberry Pi against a pilot image. The Raspberry Pi was able to correctly classify the test image with a 99.99% confidence level; based on four separate models, with three of the four models agreeing on the prediction. The same image was used for tests on the desktop and both the high performance desktop and the Raspberry Pi 3 B+ were capable of providing the correct prediction and same confidence level. The only difference was the use of four models instead of twelve, to minimize the run time required and counter issues of not enough memory for twelve models.

```
1 of 4 completed
densenet_model_ex-170_acc-0.871000.h5
2 of 4 completed
squeezenet_model_ex-001_acc-0.100000.h5
3 of 4 completed
resnet_model_ex-097_acc-0.852000.h5
4 of 4 completed


Filename:       pilotImage.jpg
Prediction:     pilot
Probability:    99.99999602635701
Voted by 3 of 4 models
(tensorflow35) root@raspberrypi:/home/pi/Downloads/imageai#
```

Figure 11: Single Image Prediction on a Raspberry Pi

During execution the Raspberry Pi was consistently over 90% memory utilization after running through five of the twelve models. Shortly into the sixth model assessment the python script crashed due to not enough memory available. Based on this limitation, the Raspberry Pi was configured to only use the top model from each algorithm instead of top three models per algorithm. All twelve models were ran individually and manually combined to verify only four models could perform as accurately as twelve. The manual combination resulted in nine of the twelve models agreeing with pilot for the prediction with a confidence level of 98.1%. The four model implementation produced the same correct profession prediction, but with an increased confidence level at 99.99%. Since the accuracy and performance increased the Raspberry Pi implementation was altered to the four model design.

The Raspberry Pi storage requirements grow at a rate of 12 KBs per image classified with a constant model storage rate of 205 MBs for four models. These requirements are portable to any edge node device used. The Raspberry Pi takes an average of five minutes to classify a new image and about 600 MB of memory for each image classification. The time to classify is dependent on the hardware used, were the Raspberry Pi takes five minutes per image the desktop takes only three minutes per image.

B. Bulk Image Classification

During the bulk image processing, the scripts ingested two thousand images, evenly distributed between ten different professions. All the images received a profession prediction from all twelve models. The Squeezenet models

consistently produced the wrong predictions, with only 10% of the predictions being correct. After further review, the model Squeezenet was only able to correctly predict a single profession. The Densenet and Inceptionv3 models all performed with an 87% accuracy and the Resnet algorithm performed slightly worse at 85%. Individually the models produced accurate results, but when implementing the voting schemes the results improved.

Table II depicts the results of two different automated ensemble methods and a third with manual intervention. Implementing simple majority voting, also known as bagging, with a confidence level threshold resulted in an 88% accuracy for predictions. The minimum required threshold eliminated the Squeezenet predictions from the voting. Where some models performed poorly for some professions other models performed strongly, resulting in increased correct predictions.

The second ensemble method involved expanding on the bagging algorithm and incorporating the confidence levels into the vote determination. Implementing this design improvement resulted in a one percent accuracy increase over the simple majority voting scheme, at 89% accuracy when averaging the predictions from all 2000 image predictions.

Table II: Training Images Classification

Ensemble Results			
Ensemble Method	Positively Predicted	Total Images	Percent Accuracy
Majority Rule	1764	2000	88%
Weighted Vote	1777	2000	89%
Weighted with Object Recognition	1807	2000	90%

Part of the future recommended work is incorporating object recognition to the predictions. The final row of Table II shows the improvement of doing manual object recognition for the waiter and chef professions. For manual recognition a tray or check book was searched for in the waiter images and a chef hat or side pocket thermometer for the chef images. The manual searching resulted in an improvement of one percent in accuracy over the weighted voting. Table III extracts the results for just the waiter and chef images. Adding object recognition gave an 8% accuracy improvement when looking solely at the chef and waiter images.

Table III: Training Images Classification

Object Recognition Result				
Prof.	Weighted Vote		Object Recognition	
	Positively Predicted	Percent Accuracy	Positively Predicted	Percent Accuracy
Chef	165 / 200	83%	174 / 200	87%
Waiter	148 / 200	74%	169 / 200	85%
Chef & Waiter	313 / 400	78%	343 / 400	86%

C. Additional Image Test

To verify the implemented algorithms additional police and firefighter images were chosen from Google searches and evaluated through the prediction script. These images were all classified correctly with over 98% certainty with at least six models agreeing on the vote. This test was conducted using the simple majority voting method. Each image was run multiple times against the same models to ensure the results were consistent each time.

An additional test of five new chef and five new waiter images were chosen from Google searches and evaluated through the prediction script. Eight of the ten new images classified correctly, at a rate of 80%. The majority of the correctly identified images had a certainty of over 90%. These images were also classified using the simple majority voting method.

VI. Hypothesis Results

This section discusses how the implemented design addresses the four hypothesizes discussed in the introductory section. The hypotheses are also listed below:

1. Utilizing a knowledge base, can a DSS be implemented to minimize the time and space requirements, while maximizing the accuracy of the suggested solutions?
2. How can a knowledge base be implemented to improve the overall accuracy of the system?
3. Is a DSS able to provide solution justification to the user, enabling users to have trust in the answers being provided?
4. Is it possible to improve rule evolution without needing to store locally all previous cases for doing rule re-validations?

A. Hypothesis 1

The implemented design does not decrease the time complexity compared to using a traditional single model process. However, by implementing a real-time image classifier on an endpoint node and the model generator on a remote server the space complexity does improve. The thought behind this is to remove heavy storage requirements from endpoints that typically have minimal resources, while the remote server typically has excesses resources.

B. Hypothesis 2

The implementation of a voting scheme with multiple algorithms used for model generation has allowed an improvement in prediction accuracy on average. Individually the models perform worse than when all models are used in a voting scheme.

C. Hypothesis 3

The voting scheme provides additional confidence in the provided result while increasing accuracy. The system provides the number of models agreeing with a prediction to assist in providing the user with greater confidence in the

result. The system also provides the probability the system believes in the response provided.

D. Hypothesis 4

The feedback loop introduces manually verified predicted images and adds those images to the pool for future model generation to enhance the model evolution. By improving the model evolution, predictions improve accuracy and allow for the system to handle input changes over time. For example, as a profession's physical characteristics evolve the system will evolve as well.

To validate the feedback loop for incremental evolution nine separate simulations was ran. Nine sets of models were generated using different training set sizes ranging from one thousand to nine thousand in one thousand increments. After the models were generated the same five hundred images were classified against the top twelve models from each model set. Table IV shows the simulations with the results. The results did not show a perfect upwards curve for accuracy, but did show that as more images were added to the training set the accuracy did improve. From this test the accuracy went from 78% at one thousand training images to 84% at nine thousand training images. The model set with one thousand images for training was an anomaly at a higher accuracy rate than the later model generations. This anomaly could be explained by the subset of images used in training just being an ideal set of images compared to the rest of image subsets.

Table IV: Model Evolution Results

Model Evolution Results				
Training Images	Positively Predicted	Incorrectly Predicted	Total Processed	Percent Accuracy
1000	391	109	500	78%
2000	316	184	500	63%
3000	349	151	500	70%
4000	360	140	500	72%
5000	381	119	500	76%
6000	399	101	500	80%
7000	400	100	500	80%
8000	424	76	500	85%
9000	421	79	500	84%

VII. CONCLUSION

Throughout this project, there were limitations to development based on the hardware available. For future development in this area, additional work should include different types of hardware platforms. The algorithms were implemented to handle both low and high-performance hardware. Implementing the system on hardware like the Nvidia 2080 Ti should improve model generation because of the additional memory and CUDA cores, allowing for improved accuracy for profession predictions. Beyond improving the specific hardware available, work with a distributed computing system should be researched.

A distributed system, shown in Figure 12, would grant a significant amount of computing power beyond what a single system would provide. A considerable improvement a distributed system would bring is a level of fault tolerance. A single system, similar to the one used in this project, has mostly all single points of failure. A distributed system would relieve the issue with single points of failure. During this project, using the Raspberry Pi attempted to simulate half the system in a distributed environment by using the Raspberry Pi as a low-powered endpoint node. The Raspberry Pi can be combined with an onboard camera to provide live image recognition as well.

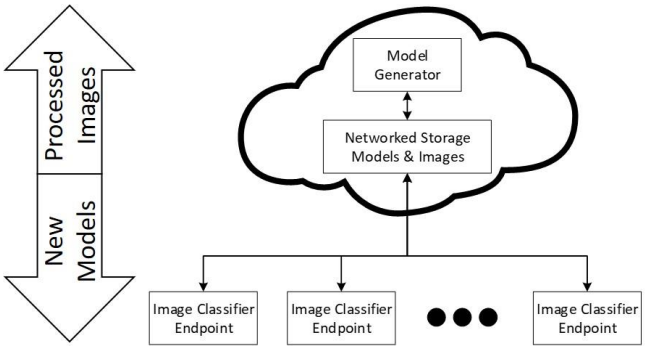


Figure 12: Distributed Network

This system's edge nodes are scalable based on the quality of the back haul bandwidth from the edge node to the centralized repository. Each individual edge node acts autonomously, with no knowledge of the other edge nodes. As long as the centralized repository has enough resources to handle obtaining all the images from the edge nodes, the generation and distribution of models then the overall system can easily scale.

Incorporating object recognition, visual shown in Figure 13, into this design would also improve the overall performance. As discussed in the results section, object recognition improved the predictions of the chef and waiter professions by 8%. This was done with only four objects manually identified, using additional objects would improve the accuracy even more.

Figure 13: Object Recognition Example [16]

Figure 14 depicts the image classification flow when incorporating object recognition. Each image is classified by twelve models using a whole image classification method. The twelve predictions are then combined through the weighted majority voting algorithm. Separately, the image is analyzed for object recognition, with the results compared against a repository of known objects linked to professions. The result from the object recognition and whole image classification is compared, if the results are the same then the system will export the result. However, if the comparison shows a disagreement then both results are evaluated for their confidence to determine what prediction to make.

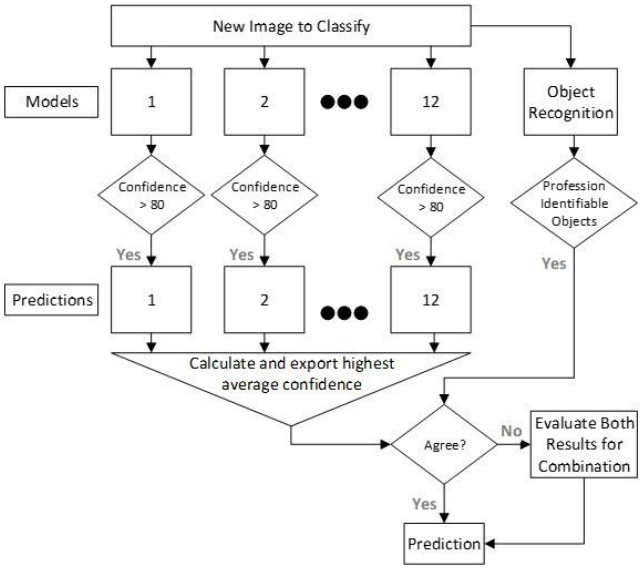


Figure 14: Object Recognition with Voting

Throughout this project machine learning algorithms and applications were reviewed to determine what improvements could be made to enhance the field. One of the greatest areas needing improvement was providing users with justification and confidence in the predictions a system is providing. This project attempted to use multiple machine learning algorithms in a voting scheme to increase the confidence level in the predictions, while also improving the accuracy of the predictions.

By using predictions as a feedback loop into the model generator, the system attempted to improve predictions over time. Improvement of the knowledge evolution is crucial for a system operating for any length of time. For instance, with the profession application, police officers over the last hundred years have evolved through their uniforms. If a system were generated using photos of police from the 1920's, then the system would have a difficult time providing correct predictions of present-day police.

With improved knowledge evolution, accuracy improvements become possible. The accuracy of the system was improved by implementing a voting scheme and a confidence threshold to drop low confident predictions. Specific algorithms showed higher performance against certain professions and were able to counteract lower-performing algorithms.

To improve the efficiency of the system, the space complexity was improved with a slightly worse time complexity. By implementing a server and client system, the size complexity was greatly reduced at the endpoint node while maintaining typical size complexity at the server-side. The client-side system only needed to download models from the server when available. Otherwise, the client can perform predictions uninterrupted. However, since the overall system utilizes the integration of open source elements there is minimal control over the inner workings of the libraries affecting size and complexity.

Overall, the implemented system addresses all hypotheses originally made by implementing common, open-source software in an untraditional manner. The delivered system from this project is capable of predicting a person's profession solely based on a single image of them, in a manner and speed humans would not be capable of achieving.

REFERENCES

[1] D. Prairie and P. Fortier, "Improve Operations of Real-Time Image Classification Utilizing Machine Learning and Knowledge Evolution", IARIA Data Analytics, 2019.

[2] A. Saxena, “Knowledge-Based Architecture for Integrated Condition Based Maintenance of Engineering Sysems," Georgia Institute of Technology, Tech. Rep., 2007, Accessed: Aug. 2019. [Online]. Available: https://smartech.gatech.edu/handle/1853/16125

[3] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, W. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty, "Building Watson: An Overview of the DeepQA Project," AI Magazine, 2010, Accessed: Aug. 2019. [Online]. Available: https://www.researchgate.net/publication/220605292 Building Watson An Overview of the DeepQA Project

[4] Google, "Tensor Flow" Google, [Online]. Available: https://www.tensorflow.org/. [Accessed Apr. 20, 2019]

[5] D. Aha, D. Kibler and M. Albert, "Instance-Based Learning Algorithms," Machine Learning, vol. 6, no. 1, pp. 37-66, 1991.

[6] F. Kittler and Josef; Roli, “Multiple Classifer Systems," in First International Workshop, MCS 2000.

[7] V. Smolyakov, “Ensemble Learning to Improve Machine Learning Results,"Statsbot, 2017, Accessed: Sep. 2019. [Online]. Available: https://blog.statsbot.co/ensemble-learning-d1dcd548e936

[8] F. P. R. E. Network, “Chances Increasing That Tropical Storm Dorian Will A ect Parts of Florida This Weekend," WUSF News, 2019, Accessed: Oct. 2019. [Online]. Available: https://wusfnews.wusf.usf.edu/post/chances-increasing-tropical-storm-dorian-will-a ect-parts-florida-weekend

[9] P. Domingos, “Bayesian Averaging of Classifiers and the Overfitting Problem," University of Washington, Seattle, Tech. Rep., 2002.

[10] J. D'Souza, “A Quick Guide to Boosting in ML," GreyAtom, 2018, Accessed: Sep. 2019. [Online]. Available: https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5

[11] “Raspberry Pi 3 Model B+," Raspberry Pi Foundation, Tech. Rep., 2019, Accessed: Aug. 2019. [Online]. Available: https://www.raspberrypi.org/

[12] DeepQuest AI, "Official English Documentation for ImageAI!" DeepQuest AI. [Online]. Available: https://imageai.readthedocs.io/en/latest/. [Accessed: Feb. 11, 2019].

[13] S. Jain, "How to easily Detect Objects with Deep Learning on Raspberry Pi", medium.com, Mar. 20, 2018. [Online]. Available: https://medium.com/nanonets/how-to-easily-detect-objects-with-deep-learning-on-raspberrypi-225f29635c74. [Accessed May. 11, 2019].

[14] M. Olafenwa, "Custom Training," ImageAI, Tech. Rep., 2019, Accessed: Sep. 2019. [Online]. Available: https://github.com/OlafenwaMoses/ImageAI/blob/master/imageai

[15] M. Olafenwa and J. Olafenwa, "ImageAI," ImageAI, Tech. Rep., 2019, Accessed: Sep. 2019. [Online]. Available: http://imageai.org/

[16] M. Olafenwa, "IdenProf Datasheet" Olafenwa, [Online]. Available: https://github.com/OlafenwaMoses. [Accessed Mar. 16, 2019]

[17] D. Galeon, "Paging Dr. Watson," 28 October 2016. [Online]. Available: https://futurism.com/ibms-watson-ai-recommends-same-treatment-as-doctors-in-99-of-cancer-cases/. [Accessed 22 February 2019].

[18] Dtex Systems, "The Hidden Security Threat," Dtex Systems, 2016. [Online]. Available: https://dtexsystems.com/portfolio-items/infographic-findings-from-the-2016-costs-of-insider-threats-report/. [Accessed 21 March 2019].

[19] Q. Althebyan and B. Panda, "A Knwoledge-Base Model for Insider Threat Prediction," Proceedings of the 2007 IEEE Workshop on Information Assurance, vol. June, pp. 20-22, 2007.

[20] P. Bakkum and K. Skadron, "Accelerating SQL Database Operations on a CPU with CUDA," University of Virginia, Charlottesville, 2010.

[21] J. Jean, G. Dong, H. Zhang, X. Guo, and B. Zhang, "Query Processing with An FPGA Coprocessor Board," in Proceedings of the International Conference on Engineering and Reconfigurable Systems and Algorithms, 2001.

[22] Martín Abadi et al, "TensorFlow: A System for Large-Scale Machine Learning," in 12th USENIX Symposium on Operating Systems Design, Savannah, 2016.

[23] D. Patil and P. Jayantrao. "Malicious URLs Detection Using Decision Tree Classifiers and Majority Voting Technique." Cybernetics and Information Technologies. vol. 18. no. 1, pp. 11-29. 10.2478/cait-2018-0002.

[24] C. Nichols, "How many flights come in and out of LAX every day?," Los Angeles Magazine, 1 May 2011. [Online]. Available: http://www.lamag.com/askchris/how-many-flights-come-in-and-out-of-lax-every1/. [Accessed Mar. 20, 2018].

[25] Homeland Security, "Combating the Insider Threat," Homeland Security, 2014.

Industry Case Study: Design Antipatterns in Actual Implementations

Understanding and Correcting Common Integration Design and Database Management Oversights

Mihaela Iridon
Cândea LLC
Dallas, TX, USA
e-mail: iridon.mihaela@gmail.com

Abstract—**Prototyping integration points with external systems and new technologies is an excellent starting point for validating certain design aspects but turning that into a complete enterprise solution goes far beyond implementing a working passthrough prototype. In some instances, the focus on functional features and tight deadlines lead to inadequate attention placed on non-functional system attributes, such as scalability, extensibility, performance, etc. Many design guidelines, best practices, and principles have been established, and antipatterns were identified and explained at length. Yet, it is not uncommon to encounter actual implementations suffering from deficiencies prescribed by these antipatterns. The first part of this paper discusses Leaky Abstractions, Mixing Concerns, and Vendor Lock-in antipatterns, as some of the more frequent offenders in case of system integration design. Ensuing problems such as the lack of proper structural and behavioral abstractions are revealed, along with potential solutions aiming to avoid costly consequences due to integration instability, constrained system evolution, and poor testability. The second half of this industry case study shows how unsuitable technology and tooling choices for database design, source code, and release management can lead to a systemic incoherence of the data layer models and artifacts, and implicitly to painful database management and deployment strategies. Raising awareness about certain design and technological challenges is what this paper aims to achieve.**

Keywords-integration models; design antipatterns; leaky abstractions; database management.

I. INTRODUCTION

Translating business needs into technical design artifacts and choosing the right technologies and tools, demands a thorough understanding of the business domain as well as solid technical skills. Proper analysis, design, and modeling of functional and non-functional system requirements is only the first step. A deep understanding of design principles and patterns, experience with a variety of technologies, and excellent skills in quick prototyping are vital. Although conceptual or high-level design is in principle technology-agnostic, ultimately specific frameworks, tools, Application Programming Interfaces (APIs), and platforms must be chosen [1]. Together they enable the translation of the design artifacts into a well-functioning, efficient, extensible, and maintainable software system [2].

Designing a solution that targets multi-system integration increases the difficulty and complexity of the design and prototyping tasks considerably, bringing additional concerns into focus. Identifying integration boundaries and how data and behavior should flow between different components and sub-systems, maintaining stable yet extensible integration boundaries, and ensuring system testability, are just a few of such concerns. This paper intends to outline a few design challenges that are not always properly addressed during the early stages of a project and which can quickly lead to brittle integration implementations and substantial technical debt.

A few recognized design antipatterns and variations thereof are explained here, including concrete examples from actual integration implementations as encountered on various industry projects. Solutions to refactor and resolve these design deficiencies and issues are recommended as well.

Section II presents a simplified perspective of a typical system integration problem. It explains a few general-purpose integration concerns and goals, and how these can help to guide the design of the overall integration solution topology and the underlying componentization boundaries.

Section III will address architectural and integration modeling concerns, focusing on a couple of design antipatterns. The structural aspects discussed in this section range from low granularity models (i.e., data types which support the exchange of data between systems) to large-grained architectural models (i.e., system layers and components). The consequences of designing improper layers and levels of abstractions are outlined, followed by recommendations on how to avoid such pitfalls by refactoring the design accordingly.

In Section IV, antipatterns covered in Section III are extended to the design of the data models and relational databases, also discussing the ability to customize external open-sourced systems that participate in the integration. Additional antipatterns are discussed, the problems behind them, as well as potential solutions that can overcome them.

In Section V, the focus is shifted to the management and delivery of the data layer components and artifacts, as databases are an integration concern that goes beyond the data exchanged between the application tier and the data tier. This section intends to explain how the choice of tools and frameworks can have a significant impact on the overall realization, management, and delivery of a robust and consistent integration solution.

Finally, Section VI summarizes the integration design and database management concerns and issues and the accompanying recommendations presented in this paper.

II. High-Level Overview of General Integration Objectives

From an architectural perspective, a given system that realizes a variety of features of its own may be designed around one or perhaps a combination of architectural styles, such as N-layered, service orientation, component-based, etc. However, when certain features rely on services or data provided by some external system or systems, employing them properly and efficiently becomes an integration requirement that must be carefully analyzed, designed, and realized.

As a general principle, a software system's quality attributes, such as extensibility, performance, testability, and maintainability, to name a few, should always be targeted by design, achieved, and continuously safeguarded. Casually bringing into the integrating system external concerns, data and behavior along with specialized technologies, libraries, frameworks, and tools, could potentially lead to a variety of problems that are difficult to resolve later.

To better understand the reasons behind this statement, Figure 1 shows an integration approach where the system on the left is integrating with a variety of targets (on the right) that provide some needed functionality. Perhaps the integration targets are added over time, one by one, as new overall features are supported. Quick, ad-hoc integration implementations, facilitated by easy access to service endpoints, APIs, and data, can lead to patchy and brittle solutions, where components from different layers of the current system become riddled with - and directly dependent on - the data, behavior, and technologies of the targeted systems. Furthermore, in some cases, even data and behavior of the integration system may leak into the external systems, if these are accessible for customizations, for example. This bleed of concerns and technology between systems is depicted by the various tiny geometric shapes in Figure 1.

With such an approach, future updates to the integration dependencies (shown as hashed geometric shapes) involve code changes throughout the integrating system, risking the overall system's integration stability, as well as potentially its performance, scalability, testability, and evolution.

Ideally, proper design of the integration points would identify new component(s) where integration concerns would be bounded to – as shown in Figure 2 and discussed in [1]. Features, data, and functionality imported from external systems would be exposed to the integrating system via interfaces/contracts that are vendor- and technology-neutral.

Adding such a layer of abstraction (denoted here as the Integration Layer) around the integration points will not only ensure a robust integration solution, but also the ability to easily swap targeted platforms in case of a product replacement (avoiding vendor lock-in), or for independent component and load testing of the integrating system, where the integration targets' features are simulated or mocked.

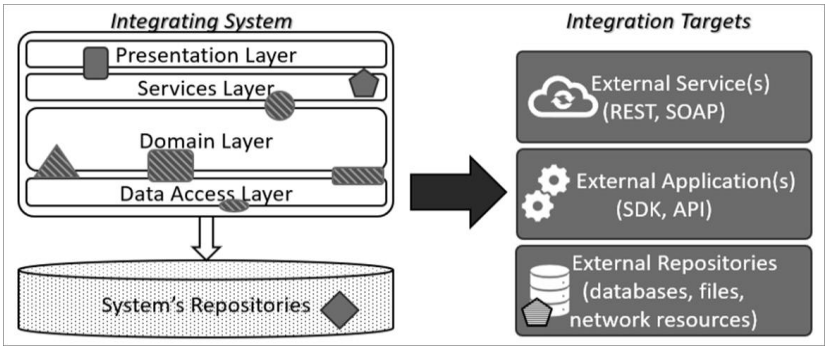


Figure 1. An unsuitable integration solution with external system concerns and technology bleeding into the integrating system (left)

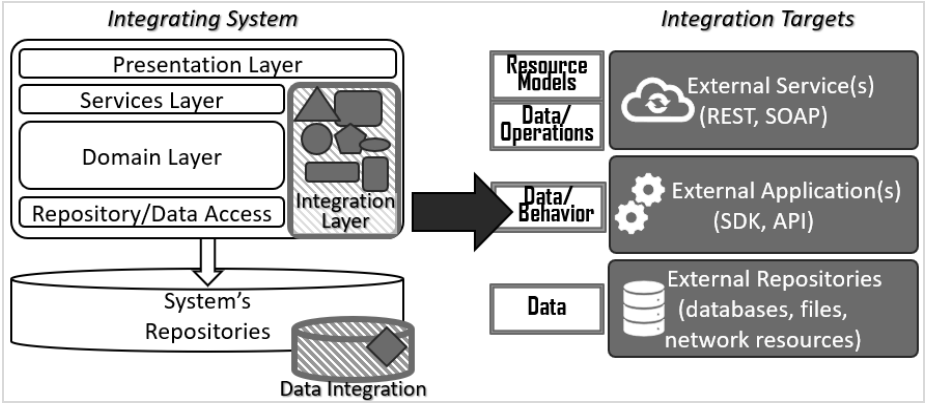


Figure 2. A fitting integration solution with a well-defined integration boundary that isolates external concerns from the rest of the system

III. TIGHT INTEGRATION: LEAKY ABSTRACTIONS AND VENDOR LOCK-IN

A. The Problem Definition

Let us consider some defined business requirements for building a software system targeting to integrate with - and consume - a third-party service. The exposed data transport models, e.g., REpresentational State Transfer (REST) models or Simple Object Access Protocol (SOAP) data contracts, are already defined, maintained, and versioned by some external vendor or entity (the service provider). Note that this scenario can easily be extended further, to integrations with an arbitrary system by means of some third-party APIs that expose specific behavior and data structures.

Focusing on the data structures rather than behavior, once the service model proxies have been generated via some automation, they tend to become part of the design artifacts for the rest of the system. Their use extends beyond the point where they are needed to exchange data with the external application. These models will percolate throughout the various layers and components of the integrating system. It is not unusual to see development efforts proceed around them, with application and business logic rapidly building on top of these data types. Development costs and tight deadlines, and sometimes the lack of design time and/or technical expertise, are the main reasons leading to this undesirable outcome.

Models exposed by external vendors were not designed with the actual needs of other/integrating systems in mind. External models are characterized by potentially complex shapes (width: number of exposed attributes or properties; depth: composition hierarchy). They cater to most integration needs ("one size fits all"), so they tend to be composed of an exhaustive set of elements to be utilized as needed.

Moreover, allowing these structural characteristics to seep into the application logic layer, beyond the component that constitutes the integration boundary, introduces adverse and unnecessary dependencies to external concerns. Therefore, the system is now exposed to structural instability and will require a constant need to adapt whenever these externally derived models will change. The integration boundary is no longer a crisp and well-defined layer that can isolate and absorb all changes to the external systems – speaking from a data integration perspective.

B. The Antipatterns

The lack of proper structural abstractions and allowing integration concerns to infiltrate into the integrating system is a costly design pitfall and is in fact a variation of the "Leaky Abstractions" problem – as originally defined by Joel Spolsky in 2002 [3]. Such deficient abstractions can be identified not only relative to structural models, but also to behavioral models, which could expose the underlying functional details of the software components to integrate with. This will inevitably lead to increased complexity of the current system, jeopardizing its extensibility and its ability to evolve and to be tested independently. Ultimately this results in a tightly coupled integration between the two systems (with strong dependencies on the target of the integration).

Another perspective or consequence of the problem described is an imposing reliance on vendor-specific technologies, their libraries, and even implementations. This problem is also known as the "Vendor Lock-In" antipattern [4]. External system upgrades will necessitate system-wide changes and constant adjustments on the integration side and will impact the overall stability of the system and the integration solution itself.

Examples range from adopting dedicated libraries for various cross-cutting concerns (logging, caching, etc.) to domain-specific technologies (telephony, finance, insurance, etc.). Vendors will encourage integrators to infuse their

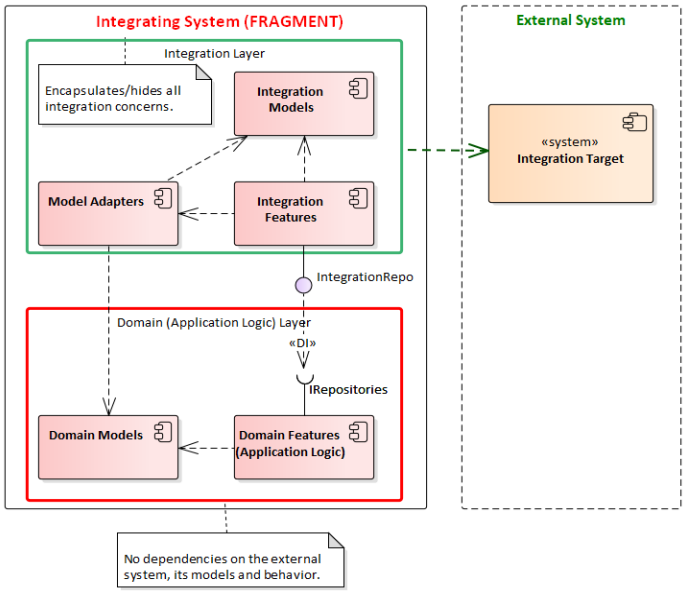


Figure 3. Integration components and the Integration Layer (Adapter) isolating and decoupling the integrating system from the external system

specialized technology everywhere, leading to entire (sub)systems taking on pervasive dependencies on their technologies, making it difficult to isolate or replace it. Such third party-entwined architectures must and can be avoided with added effort during the design phase, as described next.

C. The Solution

To avoid such scenarios, the design must unambiguously identify the integration boundary and define custom integration models that abstract away any and all structural and behavioral details related to the system targeted for integration. This architectural approach is exemplified in the component diagram in Figure 3. The integration layer should also hide the underlying technology (REST vs. SOAP, message bus vs sockets, etc.) to avoid tight and unnecessary dependencies. An example of defining canonical models based on the "ubiquitous" integration language in case of multi-system integration is presented in "Enterprise Integration Modeling" [5].

Based on the author's experience, designing proper model abstractions proved extremely useful in the case of building custom integrations with real-time systems. For example, Session Initiation Protocol (SIP) soft switches used in telecommunications networks, such as those from Genesys, the leader in customer experience, pertaining to contact center technology (call routing and handling, predictive dialing, multimedia interactions, etc.). In this case, an extensive array of data types, requests, events, etc., are made available to integrators as part of the Genesys Platform SDKs [6]. These facilitate communication with the Genesys application suite – which in turn enables integration with telephony systems, switches, IVR systems, etc. Most of these data types are very complex and heavy, and introduce acute dependencies on the underlying platform, exposing many implementation details as well. Employing code generation and metadata inspection via reflection, for example, simpler connection-less models were designed to mimic and expose only the needed structural details and are currently used in several production systems. Furthermore, defining and realizing the proper architectural isolation layers will ultimately provide independence from vendor-specific platforms for the rest of the system. For example, considering the integration scenario mentioned above using Genesys' Platform SDK shown in Figure 4, recently the company (Genesys) has been pushing for a new approach to integrate with their systems, specifically using the Genesys Web Services (GWS) [7], a RESTful API. From an integration viewpoint, this substitution is practically equivalent to switching to a different vendor, as the two integration facilities are based on different technologies (web calls versus direct socket connections) and using completely different models, from both a structural model perspective as well as behavioral and consumption views.

Building an explicit and clean integration layer as shown earlier in Figure 3, when dealing with such a significant change (vendor or technology replacement), implementation

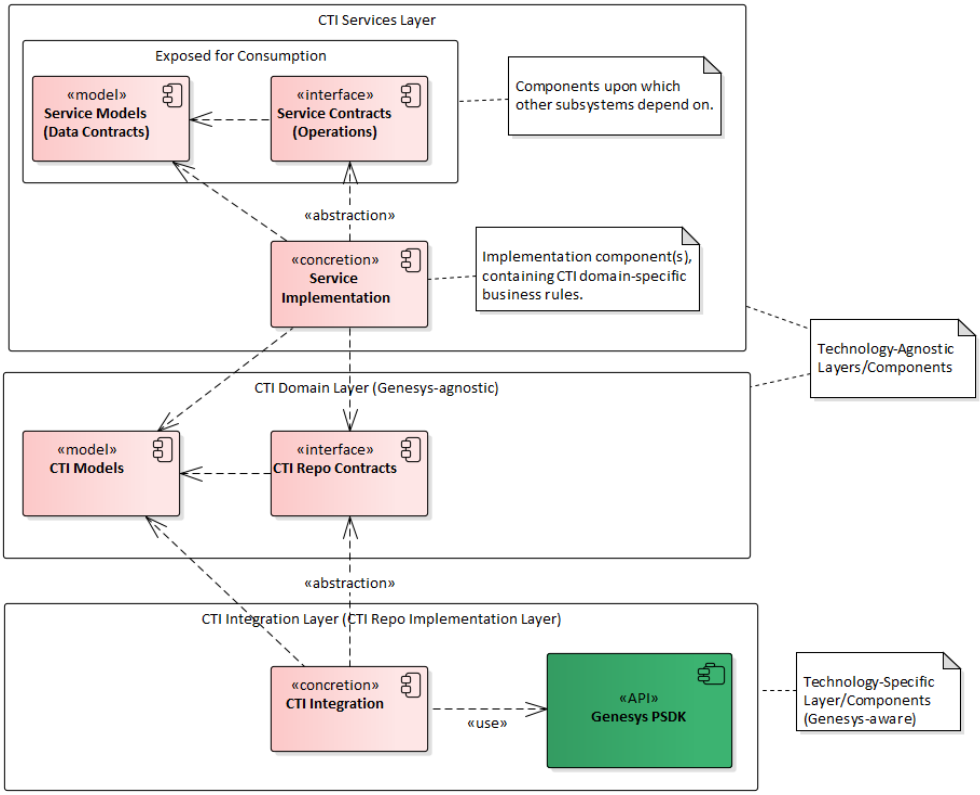


Figure 4. A sample layered architecture for exposing Computer Telephony Integration (CTI) features via integration with Genesys Platform SDK

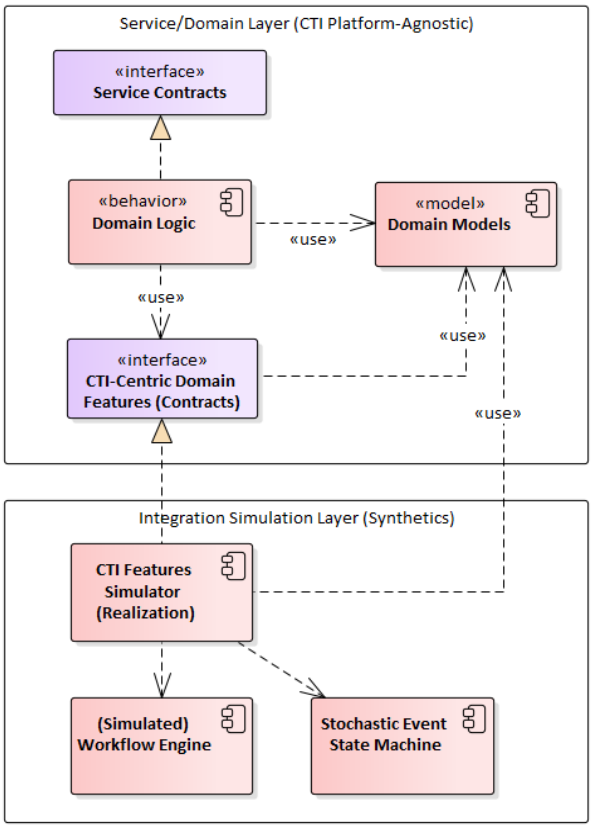


Figure 5. A concrete example of an integration architecture where the integration layer is replaced by components that simulate the integration target's functionality for testing purposes

adjustments will be isolated to this adapter layer without any impact on the business domain layer of the integrating system (assuming similar data and functionality). This includes the specifics of the technology used to communicate between the two systems.

Finally, it is noteworthy that four out of the five SOLID design principles [8] substantiate and drive towards the proposed solution:

- Single Responsibility (SRP), from the component and layering perspective,
- Open-Closed, to avoid changing the underlying implementation every time the integration endpoints change,
- Interface Segregation, exposing only the necessary data types for consumption by the business logic layer,
- Dependency Inversion, where the Domain does not directly depend on the external system, its data and behavior, but rather on abstractions – the repository contracts realized by the integration layer.

D. *Added Architectural Benefit*

Proper design and isolation of the integration components and the use of interfaces and model adapters will enable adequate testing of the custom system without demanding the availability of the external system for integration testing until most defects within the custom system are resolved.

Furthermore, this design approach supports building synthetics that simulate or mock the data and behavior of the external system, providing the means to prototype and test the integration points and functional use cases. This is exemplified in Figure 5, describing at high level a real implementation of a simulation subsystem intended to synthesize the behavior of Genesys' Statistics Server employed in a concrete integration solution.

Even if only a reduced set of features is synthesized, deferring the needs for actual integration testing can be cost-effective, especially in situations where the external system is a shared resource, perhaps expensive to manage and to access in general. Employing Dependency Injection (DI) [9], either the real or the mock implementation of the integration contracts can be injected into the Domain layer, making it easy to swap between the two implementations.

IV. Data Tier Design and Data Access Anitpatterns

One of the most common system integration use cases for many enterprise applications is related to data persistence and access. Integration with (relational) databases that are either part of the custom system or accessible (co-located) components of a third-party system is a pervasive requirement, whether the data tier is needed for storing configuration data, audit/logging, security-related aspects, or to support concrete operational or reporting needs.

This section focuses on several issues related to both database design as well as accessing the data itself.

A. *'Inverted' Leaky Abstractions in Data Integrations*

1) *The Problem*

The previous section discussed Leaky Abstractions that result from allowing third-party concerns to infiltrate custom systems when designing and implementing an integration solution. The directionality of the "leak", as described earlier, is from the external system into the current one. However, it is also possible to encounter the reverse scenario when the integration target is open or transparent to the integrators who then take advantage of this fact to develop and apply their own customizations onto the external system.

Here are two examples:

(a) An Original Equipment Manufacturer (OEM) and/or White Label license of the external system is available to integrators, including access to source code for additional customization and integration options.

(b) The external system contains database(s) accessible to the integrators, either deployed on premise or in a cloud environment, and is open/accessible to change.

In the first example, the same issues and solutions apply, as already discussed in the previous section, only this time from the perspective of the external system. If customization design is not executed properly, software upgrades of the open-sourced third-party system will result in continuous maintenance, or worse, breaking the custom code. Both scenarios will incur high development and system integration testing costs, among other problems.

The rest of this sub-section will focus on the second example, involving third-party databases that are accessible (i.e., open to modification) from an integration and customization perspective.

When expecting and relying on continuous upgrades and patches supplied by the vendor of the external system, it is possible that custom database artifacts (added by the integration provider) will have to be discarded and reapplied, or worse, no longer compatible with the updated system. Moreover, management of database source code targeting the customizations is more difficult if tightly dependent on the elements defined by the external entity/vendor. For example, the custom integration requirements demand two new columns on one of the third-party database tables.

Evidently, with respect to customizations of third-party components (database or otherwise), "Vendor Lock-in" is the status quo as a business-driven need and not a concern here.

2) The Solution

There are several options available and their applicability depends on concrete scenarios and business needs. Ideally, a separate, custom database could be considered, where data collected by the third party system (stored in their databases) would be Extracted, Transformed as needed, and Loaded (ETL) [10]. Detached custom data models are easy to maintain, modify, and version-control by the integration provider. Aligning with the arguments stated in Section III, this approach enforces a well-defined data integration boundary, as shown in Figure 6 below.

Allowing for independent provisioning and evolution of both data models (one provided by the external system and one specifically designed for - and consumed by- the integrating system) will lead to improved extensibility, scalability, performance, testability, and maintainability. With this approach, upgrading the external system will potentially require updating the ETL artifacts and, if needed, some enhancements to the custom database – but both activities can be done in a detached, self-contained fashion.

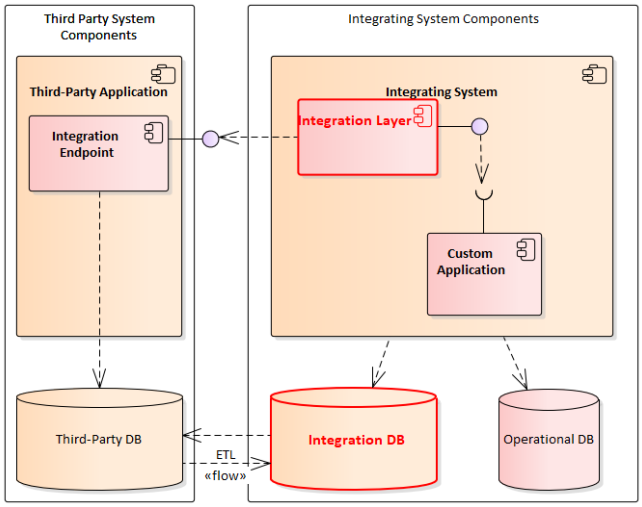


Figure 6. The integration database added to support data integration customizations and to remove direct dependencies on the third-party database

Further details regarding the management of database artifacts will be discussed later, but one noteworthy benefit here is the freedom from having to maintain (a) partial custom database artifacts (divorced from their context) and/or (b) complete external database artifacts (since the database is a self-contained software system, and should not be divided further into sub-components). The reason why maintaining select/partial database artifacts is undesirable is that from a specification perspective, a database (meaning all its defining artifacts) must be valid, consistent, and complete (as it must also be from a deployment perspective).

If database customizations *must* live in the same database as the one that is part of the external system (perhaps for performance considerations), a less optimal solution to the Inverted Leaky Abstractions (i.e., the data model), is to expend proper design effort to minimize tight dependencies and attempt to follow - as best as possible - the Open-Closed design principle at the data tier, in the context of system integration and customization.

For example, if the custom integration components require the persistence of new attributes (fields) in addition to the data captured by the external system, rather than modifying the existing third-party tables by adding new columns, association or edge tables should be considered instead, with custom data residing in new, custom tables. Custom views, parameterized or otherwise, should be designed to transform data into a ready-to-consume format (for operational, reporting, or analytical needs).

In this case, the system quality attributes mentioned earlier must however be carefully monitored, especially query performance and scalability.

On the downside, database code management will become either (a) fragmented/isolated, by extracting the custom database artifacts from the rest of the database into independent scripts, or (b) more complex, by importing the entire third-party database under source control along with the custom artifacts, in order to preserve its integrity.

Section V discusses tools that help validate the full database, warning about invalid or broken object references, binding and syntax errors, thus increasing the probability that database deployments will succeed.

B. Mixing Data Modeling Concerns

1) The Problem

Regardless of the targeted Database Management Systems (DBMS) technology, designing the conceptual and logical data models is a prerequisite to the implementation of the physical data models [11]. Beside ensuring that all data elements outlined by the business requirements are accurately represented, non-functional requirements, such as performance, scalability, multi-tenancy support, security (access to data), etc., will also shape the data architecture.

From an application perspective, the database is used to persist the state of the business processes supported by the application, i.e., operational needs, and to support analysis and reporting needs around the stored business data. The concept of Separation of Concerns (SoC) applies here as well but is often ignored or inadequately addressed. Operational versus reporting concerns are often mixed and data models

designed specifically for operational needs are used as such for reporting or analytics purposes, although these models are usually quite different, in terms of how the data is stored and how it is accessed. Yet, it is not uncommon to find a given database used both as the operational as well as the reporting database. As a direct consequence of violating SoC with respect to data modeling (both logically and physically), stability, scalability, extensibility, and performance are the main quality attributes of the system that will be impacted.

An alternate description of this problem is known as the "One Bunch of Everything" antipattern [12], qualifying it as a performance antipattern in database-driven applications, the author aptly pointing out that "treating different types of data and queries differently can significantly improve application performance and scalability."

2) The Solution

Following general data architecture guidelines, the solution is straightforward. In [13], Martin Fowler suggests the separation of operational and reporting databases and outlines the benefits of having domain logic access the operational database while also massaging (pre-computing) data in preparation for reporting needs. Extract-Transform-Load (ETL) pipelines/workflows can and should be created to move operational data into the reporting database; specifically, into custom-tailored models that cater to requirements around reporting and efficient data reads.

Existing tooling and frameworks can be employed to transform and move data efficiently, on premise or in the cloud (Azure Data Factory, Amazon AWS Glue, Matillion ETL, etc.), for data mining and analytics, for historical as well as real-time reporting needs.

C. Data Access and Leaky Abstractions

1) The Problem

It has been noted [14] that Object Relational Mapping (ORM) technologies, such as Entity Framework (EF) or Hibernate, are in fact a significant cause of data architecture bleed into the application logic, representing yet another example of the Leaky Abstractions antipattern.

Although intended to ease the access to the data tier and the data it hosts, such technologies expose underlying models and behavior to the application tier. In more acute cases – depending on its usage – it also introduces strong dependencies from the domain logic to the data shapes defined in tables, views, and table-valued functions.

Entity Framework, for example, while providing the ability to create custom mappings between these data models and the entity models, as designed, these object models are intended to be used as the main domain entities to build the actual domain logic around them. This forces a strong, intertwined yet inadequate dependency between two very different models, targeting different technologies, employed by very different programming paradigms (OO/functional such as C#.NET versus set-based such as SQL). This not only restricts the shape of the domain models, forcing constrained behavioral models to be implemented around them, but also causes data architecture changes to affect the domain and the application logic itself.

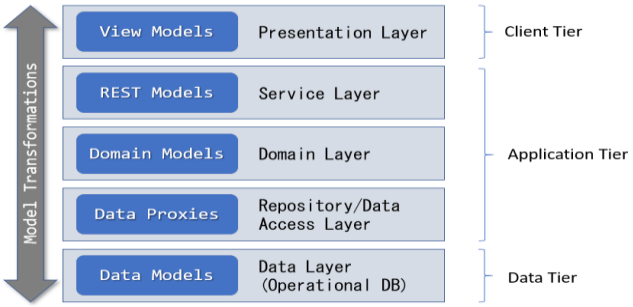


Figure 7. Layered architecture with layer-specific models and model transformations

Not surprising, Microsoft's EF Core framework in fact discourages against using a repository layer [15] (as prescribed by Evans's DDD [16]) on account that EF itself implements the repository pattern/unit of work enterprise pattern [17] – alas, leading towards a rigid and potentially brittle integration. The reason is that ORM technologies push design and development towards data access logic tangled with the domain logic by encouraging multi-purpose models (domain and data access or data proxies).

2) The Solution

Just as with the integration solution presented in Figure 3, the impact of changes to database models should be constrained to one or two components – those that make up the data access layer, and prohibited from affecting the other application layers, specifically the domain and service layers. Sharing a single model across all layers of the application places unnecessary limitations on the overall design and ultimately on the extensibility and stability of the system.

Although it is uncommon to replace the database technology altogether, sometimes it may be required to replace the *data access* technology due to performance and scalability concerns. Without a proper separation of data access from domain logic and models, such design changes targeting the lower layers of the system architecture are impractical without extensive refactoring of the application.

In a layered component-based architecture – as shown in Figure 7 above, it is easy and natural to allow each layer to define its own models (darker boxes) and provide adapters to translate from one model to another as data flows through the layers of the application by means of interfaces. Although this would seem wasteful at first sight, especially if some models hardly vary from one layer to the next, this approach offers two core benefits. It allows for independent evolution of the models, customizing them to serve very specific needs of the layer they belong to, and keeps the propagation of model changes confined to the corresponding adapter (translation) components.

In case of ORM technologies, the data access layer overlaps with the domain layer, while entity models (shown as data proxies in Figure 7) represent the actual domain models. Interestingly enough, even as ORM is recognized as a Leaky Abstraction, its use is nevertheless encouraged [18], most likely because in unsophisticated implementations, it may be able to deliver acceptable results.

However, as Bilopavlovic points out [14], ORM tools can be successfully used "if there is proper separation of concerns, proper data access layer, and competent developers who know what they are doing and really, really understand how relational databases work." Sooner or later, the inherent deficiencies of such technologies, compounded by inadequate implementations due to the lack of understanding of how the underlying technology works, will surface, in most cases under system load and/or when new features are added.

V. DATA TIER MANAGEMENT CONCERNS

Previous sections discussed antipatterns as relevant to the design of software solutions, specifically to the design of software systems integration. Bad practices and approaches can be encountered in several areas, beside design: in code implementations, in management of the code and software artifacts, in activities pertaining to DevOps, such as deployment and change management, etc.

This section will focus on inadequate practices around management of relational databases, with a detailed focus on Microsoft SQL Server relational databases, tooling and frameworks used for change management and incremental deployments, among other things.

A. Improper Management of Database Artifacts

1) The Problem

Source code, regardless of the language it is written in, is "a precious asset whose value must be protected", as Atlassian's Bitbucket web site states in their "What is version control" online tutorial [19]. All software-producing companies will employ one tool or another for version control. This allows software developers to collaborate, store (or restore/rollback) versions of the software components they build and perform code reviews, providing a single, stable "source of truth" of the software artifacts they create and release/deploy. As advocated in [20], "source files that make up the software system aren't on shared servers, hidden in folders on a laptop, or embedded in a non-versioned database." Yet, it is rather commonplace to find database implementations that are improperly managed, leading to frustration, bad deployments, making the data tier integration and overall solution delivery unreliable and difficult. There are many online articles and blogs describing such cases.

As encountered by the author, while being engaged as a solution architect and consultant on several projects at various clients, the actual data models and database artifacts were often created and delivered as *ad-hoc implementations* in some arbitrary database, hosted under some *arbitrary* Microsoft SQL Server database instance. Several teams needed these database artifacts: Development for implementation and integration, Quality Assurance for testing, Business team (domain experts and business analysts) for reporting and analytics, and DevOps for deployment. The most common process for deploying this database (fresh install or incremental) to some other environment was to generate and pass around SQL scripts when needed. In somewhat more fortunate situations, these scripts were maintained in some form of source control as SQL/text files but lacking the ability to validate them or trace the source back to the developer responsible for the actual implementation (in the original database).

So then, where does the "source of truth" for the database definition reside? How can multiple developers work on the database code without overwriting each other's changes and without being aware of the latest updates? How does the organization deliver incremental deployments to any number of target environments? When onboarding new team members, what database code should they be pointed to?

The problems derived from not having a stable, accurate, up-to-date, and complete definition of the database source code, one that is under version control and that can be validated before a deployment, are numerous, acute, and rather obvious. Just as one maintains all other application code under source control, entire solutions composed of many components, why should database implementations not follow the same standards and take advantage of the same acclaimed benefits of code well-managed?

Furthermore, when the database (source) code resides in some database, invalid object references (because someone dropped a column on a table or deleted a stored procedure) will surface only at runtime. Often, changes are made to the database post deployment, even in Production environments, changes that could potentially break the code, or which are at best confined to that environment alone, but without being retrofitted/updated back into the "source code database".

A particularly curious approach to database code management and deployment was encountered on a project that used the Fluent Migrations Framework for .NET [21], self-proclaimed as a "structured way to alter your database schema […] and an alternative to creating lots of sql scripts that have to be run manually by every developer involved." In a nutshell, the tool calls for creating a C#.NET class every time the database schema would change (one class per "migration"). These code files (admittedly, version-controlled) attributed with metadata to identify a specific database update, encapsulate two operations that describe the schema changes: one for a forward deployment ("Up") and one for rollback ("Down").

A very simple example, involving the source code of a rather trivial stored procedure, is shown in Figure 8.

With a large database, one that evolved considerably over time, with hundreds of artifacts, the number of C# migration files was astounding (thousands). Database changes were published to the target database as part of the application deployment process. Installing the database from scratch would incrementally apply every single "Up" migration specification found in these files, following the prescribed update. To maintain sanity, these source code files needed to be named such that the chronological order would be preserved when browsing through them in the development environment tool.

However, other more serious problems arise from using this framework, two of them being briefly discussed next.

```
using FluentMigrator;

namespace DatabaseMigration.Migrations
{
  [Migration(98)]
  public class M0098_CreateStProcAddNodes : Migration
  {
     public override void Up()
    {
        Execute.Sql(@"CREATE PROCEDURE [cfg].[AddNodes]
          @nodes cfg.NodeType READONLY
          AS
          BEGIN
            SET NOCOUNT ON;
            INSERT INTO cfg.Node
              (Name, Value, ValueType, CreatedBy,
               CreatedDate, UpdatedBy, UpdatedDate)
            SELECT s.Name, s.Value, s.ValueType, s.CreatedBy,
               s.CreatedDate, s.UpdatedBy, s.UpdatedDate
            FROM @nodes as s;
          END");
    }

    public override void Down()
    {
      Execute.Sql("DROP PROCEDURE [cfg].[AddNodes]");
    }
  }
}
```

Figure 8. Sample C#.NET migration code for adding a new stored procedure and rolling back the change

a) SQL code as C#.NET strings??

Say a new stored procedure must be added; the code is developed and tested from SQL Server Management Studio (SSMS) in some local deployment of the database (assuming the objects the stored procedure is referencing do not change in the interim). Next, a migration file is created, with the "Up" method containing the full (CREATE) stored procedure script, *as a C# string* passed as input argument to the "Execute.Sql" method call. A sample migration code snippet describing this scenario is shown in Figure 8.

The major and obvious problem here is the inability to validate SQL syntax and semantics and SQL object references when represented as indiscriminate plain strings, subject to typing errors.

b) No database source code??

Unless deployed on some SQL Server instance, it is impossible to even begin to understand the structure of the database, even the structure of individual objects. The data models and data logic are scattered, fragmented (across many C# files), impossible to validate (syntactically or otherwise) from where the database "source code" is stored.

Moreover, a given database object, say a table for example, can change any number of times, each change being captured in a different source file, with no unified, single view of what that table looks like, what the shape of the data is, with all its columns and corresponding types, with its keys and indexes, constraints and triggers, if any. This problem extends to all database objects, not just tables.

The data models (the source code artifacts) are practically non-existent, disjointed, difficult to comprehend, and cannot be validated until they are deployed. The result is a total and indefensible representational incoherence afflicting the most important component of a data-dependent enterprise system.

2) The Solution

There are various software tools available to address this problem. Both Microsoft and Redgate, for example, provide excellent tooling for developing relational databases, managing database artifacts under source control, facilitating change management and incremental deployment, generating manual update scripts (when automated deployment is constrained), and more.

Microsoft's SQL Server Data Tools (SSDT) [22] is a development tool, available since 2013, using the Data-Tier Application Framework (DacFx). It facilitates the design and implementation of SQL Server and Azure SQL databases, as well as database source control and incremental deployment, all integrated under the Microsoft Visual Studio development environment.

A version-controlled database project contains all distinct database objects as individual files, and it *must* compile – targeting a specific SQL Server (or Azure) database version – before it can be deployed anywhere. Developers can check out individual objects (files) to change as needed or can add new objects using the provided templates. Just as one can see the entire schema of a database in SSMS, similarly they can see and browse these objects in Visual Studio, as shown later in the development environment snapshot in Figure 9. Here, the main database project (`Config.Database`) is – like all projects in the bounding solution – subjected to building or compilation. As a result, two artifacts are being created: a managed assembly file (.dll) and a data tier application package (.dacpac) file. Both are required for actual database deployment, but it is the .dacpac that holds the actual and full database definition. It is used by the Microsoft tooling (SqlPackage.exe) employed for incremental deployments (schema updates) against targeted environments.

It is highly questionable to store Java or C# code in SQL scripts, with artifacts/classes shredded and reduced to SQL NVARCHARs, scattered in an arbitrary number of stored procedures (equal to the number of updates effected upon that class), and passed around to call other stored procedures (via EXEC statements). The reverse scenarios should be equally unacceptable. Treating the database as a proper software implementation artifact is imperative.

B. Database Development and Deployment Concerns

1) The Problem

Tools like SSDT are also capable of identifying the changes (delta) between the source and the destination database in order to create the appropriate deployment scripts, and ultimately allowing rapid and valid delivery of database changes to any environment. Quite frequently, multiple teams are involved in database development: backend developers of applications relying on persisted data as well as data migrations (ETL) and reporting developers. Bringing all teams together to follow unified and consistent database development and deployment processes can be challenging.

Furthermore, how can specialized implementations be properly designed, source-controlled, and deployed seamlessly, while keeping the two implementations (core and custom) separate but dependent solution components?

Considering a database (and hence its associated project) as part of a larger software system, as an essential component of that system, requires indeed some additional effort in designing and managing all system's artifacts under a unified solution framework. If libraries and executables are easy to group around layers and features, whether they cater to domain versus cross-cutting concerns of that system, database componentization strategies may not be straightforward. However, recognizing that even databases and their underlying objects (i.e., code) can be broken down into logical parts, will facilitate management of these artifacts and better extensibility.

To better understand this, consider a database that consists of core objects (tables, procedures, functions, etc.), perhaps part of a product line that evolves over time. Some customers may ask for certain customizations, for example, that require additional database objects to be created, specific to their business rules and models (as would be the case of custom reports that rely on custom views).

One option is to design and implement these new views directly in the targeted environment, without including them into the source-controlled database component. The database/reporting developers would separately maintain these objects, but when later the underlying tables change, the views referencing them may break, and hence the validity of the reporting component is jeopardized.

2) The Solution

Alternatively, extensions of the core database component can be created – as separate database projects, holding only these additional custom objects, with a same-database dependency setting to the main database (project). Teams can independently work on core versus custom components, both being validated (compiled) and source controlled.

Figure 9 shows such a solution, with two database projects (components), one extending the other, with the extension component, `ConfigExt.Database`, having a dependency to the main component via database reference. Then, for actual deployment, the extension database package would be used – as it contains both custom objects as well as the core database objects from the referenced component, resulting in a *full* database installation or update.

The tooling and processes described here, as already mentioned, target the Microsoft technology stack. However, similar options exist for other platforms as well, more or less effective in various areas or others, to assist with development and management of enterprise databases.

Figure 9 shows a snapshot of a solution developed under the Microsoft Visual Studio environment, with two of the 19 projects being a couple of rather trivial database projects, `Config.Database`, and its extension, `ConfigExt.Database`. Either project encapsulates an entire (yet simple) database with all its objects grouped under schemas and object type folders. The top right panel shows the same stored procedure

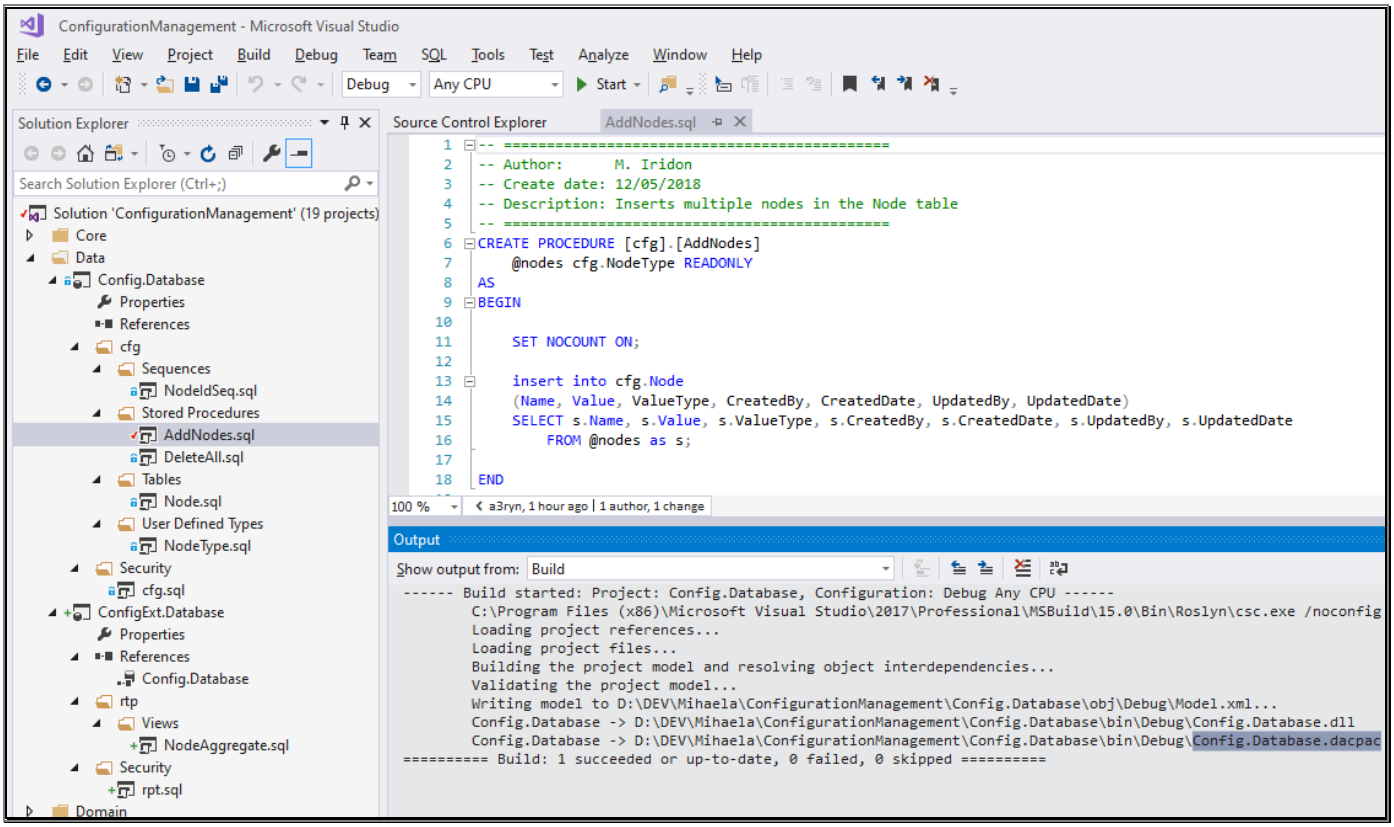


Figure 9. Database source code managed in Microsoft Visual Studio via SQL Server Data Tools.
Code files are checked in or out from a source control repository (shown on the left) as database development in is progress.

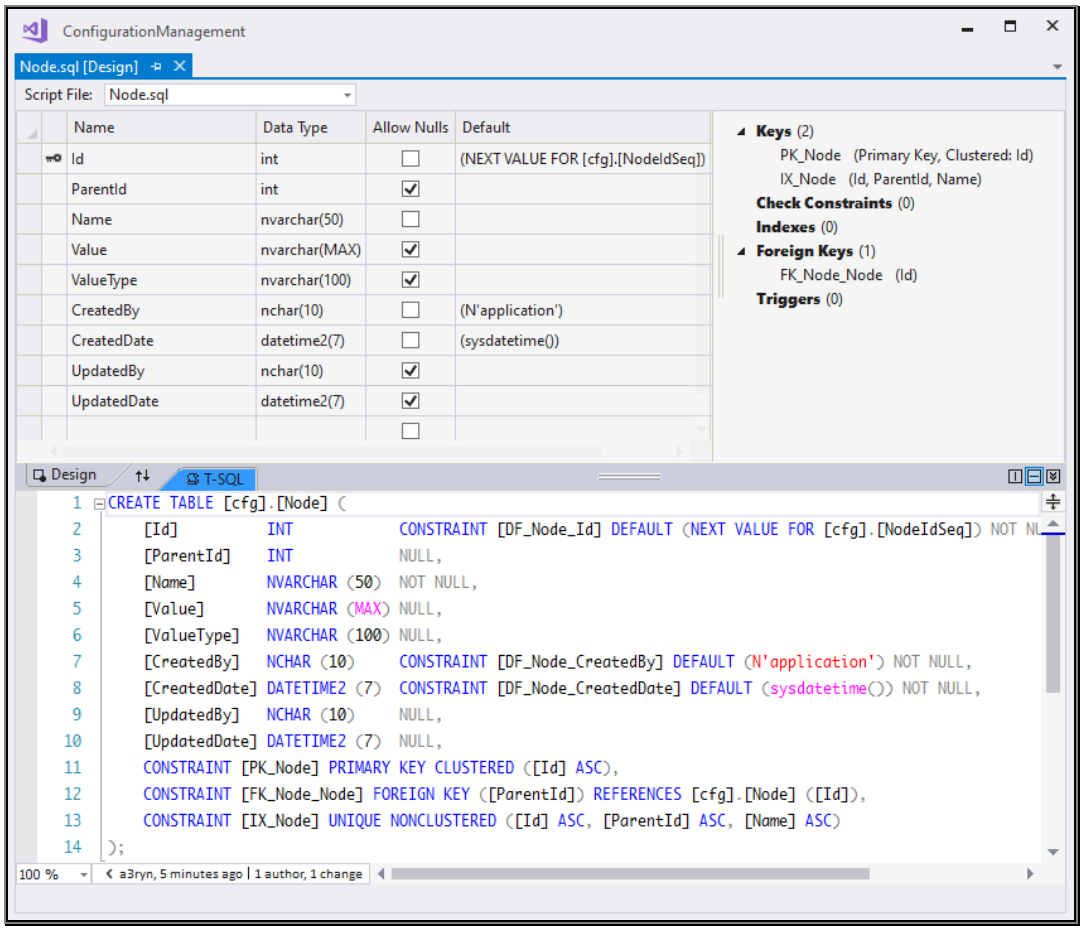


Figure 10. Database table designer (top) and script (bottom) snapshot in Microsoft Visual Studio, using SQL Server Data Tools

from earlier in Figure 8– whose source code was captured there as a C#.NET string. In contrast, here it is managed as a proper element of the database, that can be compiled (validated) and independently tracked for code changes.

The project/database compiles successfully, as shown in the bottom part of the screenshot in Figure 9. The build output artifact, i.e., the data tier application package `Config.Database.dacpac`, is highlighted.

Similarly, table objects (including indexes, constraints, etc.) can be managed in a fashion that resembles the look and feel of the table designer utility in SQL Server Management Studio. This visual design feature is captured by the top section in Figure 10. Otherwise, the scripting option (bottom) is always available, for all object types.

For all database objects, only the CREATE statement is used in all SQL source code. The tooling itself determines, at deployment/publishing time, whether CREATE or ALTER Data Description Language (DDL) statements will be required based on the delta between the concrete target database and the database source code. This greatly simplifies deployment of SQL databases against any environment, including fresh installations as well as incremental updates.

Finally, as far as employing SQL Server Data Tools and treating databases as proper software artifacts, we can enumerate below some of the key benefits that should encourage software companies to adopt SSDT, should they design and develop solutions around Microsoft's SQL Server relational databases.

To briefly summarize, here are these benefits, which should be considered perhaps also as a guiding set of objectives for any database development activity:

- ✓ Providing a unified perspective of a database,
- ✓ Validating correctness of the database definition,
- ✓ Validating completeness of a database definition,
- ✓ Providing support for version-control of the database artifacts (at the database object level),
- ✓ Allowing to perform schema comparison,
- ✓ Facilitating incremental deployments (change management), directly against a target database or via SQL scripts,
- ✓ Enabling the logical and physical componentization of databases, to facilitate the customization, extensibility, and manageability of the underlying artifacts.

VI. Conclusion

This paper aimed to raise awareness about certain design challenges that, when not addressed early and properly, will lead to deficient architectures and rigid solutions concerning various aspects of system integration, as often encountered in practice.

When the design of software systems follows some basic guidelines and principles (SOLID), the resulting architecture will allow the system to be easily built, modified, and extended. In case of system integrations and customizations, violating these principles and particularly the multi-faceted Separation of Concerns design rule, leads to unmanageable and highly complex systems that do not scale well, cannot be extended or modified easily, with tight dependencies on external components and overall brittle integration solutions.

Many design antipatterns have been catalogued and well documented; yet deficient architectures are encountered quite frequently, leading to high technical debt and unhappy stakeholders. This paper discussed "Leaky Abstractions", "Mixing Concerns", and "Vendor Lock-in" antipatterns – from the perspective of concrete industry examples, as encountered and worked on by the author.

Concrete approaches that address these problems to help refactor and realign the design according to best practices and principles were elaborated, explaining how they lead to scalable, extensible, testable, efficient, and robust integration solutions.

Relational database design and management concerns were also presented, with focus on data model design, data access practices, and management of database artifacts. The consequences of improper tooling and frameworks were briefly covered, and a technology-specific solution targeting Microsoft SQL Server databases was discussed.

Acknowledgement

I would like to thank my husband and long-time mentor, Chris Moore, for his indefatigable guidance and for sharing the extensive technical knowledge and experience he possesses and masters so adeptly.

References

[1] M. Iridon, "Industry Case Study: Design Antipatterns in Actual Implementations. Understanding and Correcting Common Integration Design Oversights," FASSI 2019: The Fifth International Conference on Fundamentals and Advances in Software Systems Integration, ISBN: : 978-1-61208-750-4, pp. 36-42, Nice, France, October, 2019.

[2] R. Martin, "Clean Architecture," Prentice Hall, 2018, ISBN-13: 978-0-13-449416-6.

[3] J. Spolsky, "The Law of Leaky Abstractions," [Online] Available from https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/ [retrieved: May, 2020].

[4] SourceMaking, Software Architecture AntiPatterns, [Online]. Available from https://sourcemaking.com/antipatterns [retrieved: May 2020].

[5] M. Iridon, "Enterprise Integration Modeling," International Journal of Advances in Software, vol 9 no 1 & 2, 2016, pp. 116-127.

[6] Genesys, "Platform SDK," [Online]. Available from https://docs.genesys.com/Documentation/PSDK [retrieved: May, 2020].

[7] Genesys, "Web Services and Applications," [Online]. Available from https://docs.genesys.com/Documentation/HTCC [retrieved: May, 2020].

[8] G. M. Hall, "Adaptive Code via C#: Agile coding with design patterns and SOLID principles (Developer Reference)," Microsoft Press, 1st Edition, 2014, ISBN-13: 978- 0735683204.

[9] M. Seemann, "Dependency Injection in .NET," Manning Publications, 1st Edition., 2011, ISBN-13: 978-1935182504.

[10] Microsoft, "Extract, Transform, and Load (ETL), " [Online]. Available from https://docs.microsoft.com/en-us/azure/architecture/data-guide/relational-data/etl [retrieved: May, 2020].

[11] G. Simsion and G. Witt, "Data Modeling Essentials," Morgan Kaufmann; 3rd edition, 2004, ISBN-13: 978-0126445510.

[12] A. Reitbauer, "Performance Anti-Patterns in Database-Driven Applications," [Online] Available from https://www.infoq.com/articles/Anti-Patterns-Alois-Reitbauer/ [retrieved: May, 2020].

[13] M. Fowler, "Reporting Database, " [Online]. Available from https://martinfowler.com/bliki/ReportingDatabase.html [retrieved: May, 2020].

[14] V. Bilopavlovic, "Can we talk about ORM Crisis?". [Online] Available from https://www.linkedin.com/pulse/can-we-talk-orm-crisis-vedran-bilopavlovi%C4%87 [retrieved: May, 2020].

[15] Jon P. Smith, "Entity Framework Core in Action," manning Publications, 2018, ISBN-13: 978-1617294563.

[16] E. Evans, "Domain-Driven Design: Tackling Complexity in the Heart of Software," 1st Edition, Prentice Hall, 2003, ISBN-13: 978-0321125217.

[17] M. Fowler, "Patterns of Enterprise Application Architecture," Addison-Wesley Professional, 2002.

[18] M. Fowler, "OrmHate," [Online]. Available from https://martinfowler.com/bliki/OrmHate.html [retrieved: May, 2020].

[19] Atlassian, "What is version control, " [Online]. Available from https://www.atlassian.com/git/tutorials/what-is-version-control [retrieved: May, 2020].

[20] P. Duvall, "Version everything," [Online]. Available from https://www.ibm.com/developerworks/library/a-devops6/ [retrieved: May, 2020].

[21] "Fluent Migrations Framework for .NET," [Online]. Available from https://fluentmigrator.github.io/ [retrieved: May, 2020].

[22] Microsoft, "SQL Server Data Tools," [Online]. Available from https://docs.microsoft.com/en-us/sql/ssdt/sql-server-data-tools [retrieved: May, 2020].

Coping with Technological Diversity by Mixing Different Architecture and Deployment Paradigms

Philipp Helle, Stefan Richter, Gerrit Schramm, and Andreas Zindel
Airbus Central R&T
Hamburg, Germany
email: {philipp.helle, gerrit.schramm, stefan.richter, andreas.zindel}@airbus.com

***Abstract*—In a time when competition and market in aviation industry drive the need to shorten development cycles especially in early phases, both automation of processes and integration of tools become important. While constraints, such as make or buy decisions or corporate Information Technology (IT) governance influence the overall tool infrastructure in different directions, microservices are a fast-rising trend in software architecting. But that does not mean that the more traditional monolithic software architecture is dead. A resulting mixed-paradigm software architecture can also be seen as an opportunity to profit from the best of both worlds. To support a newly developed complex system development approach called SCM modeling, a supporting application framework prototype is subject to development with the objective to reduce both time and resources required during product development cycles. This paper describes the software architecture styles and deployment approaches that were used in a research project at Airbus for building a prototype and discusses challenges and opportunities that were encountered. Furthermore, it describes a change in the aircraft development process to cope with the increasing complexity of products and pressure from the market to develop aircraft faster. The key process change is a deviation from the traditionally linear development approach and the inclusion of an Out-of-Cycle (OOC) development phase, where components of an aircraft are developed preemptively, out-side of a program development effort.**

Keywords–model-based systems engineering; microservices; REST; component-based development; aircraft development process.

I. Introduction

This article is a revised and extended version of the article [1] originally presented at the The Fifth International Conference on Fundamentals and Advances in Software Systems Integration (FASSI 2019).

The Microservice Architecture (MSA) is a style that has been increasingly gaining popularity in the last few years [2] and has been called "one of the fastest-rising trends in the development of enterprise applications and enterprise application landscapes" [3]. Many organizations, such as Amazon, Netflix, and the Guardian, utilize MSA to develop their applications [3].

Pursuing the notion that "Microservices aren't, and never will be, the right solution in all cases" [4], this paper describes the architecture and development approach that was used in a research project at Airbus for building a prototype application framework for Model-based Systems Engineering (MBSE). According to the International Council on Systems Engineering (INCOSE), "MBSE is the formalized application of modeling to support system requirements, design, analysis, verification and validation, beginning in the conceptual design phase and continuing throughout development and later life cycle phases" [5]. This framework does not rely on a single paradigm but instead mixes different paradigms, viz. architecture patterns and deployment approaches, to achieve the overall goals: agility, flexibility and scalability during development and deployment of a complex enterprise application landscape.

This paper is structured as follows: Section II describes the modeling method that the built prototype MBSE framework is supposed to support. Section III provides background information regarding the different enterprise application architecture paradigms. Section IV explains the IT infrastructure, in which the framework is deployed and Section V describes how and what features have been implemented in the prototype. Section VI will present the usage of the described framework using a small case study. Section VII discusses advantages and disadvantages of the mixed-paradigm approach. Section VIII talks about the ongoing and future improvement effort before section IX wraps everything up with a conclusion.

II. Smart Component Model (SCM) modeling method

In [6], we provide a detailed account of the newly developed MBSE paradigm, called SCM modeling, which is rooted in a proposed change in the aircraft development process to include an OOC component development phase, in which components of an aircraft are developed independently of the traditional linear development process. These components are then adapted to the specific needs of a program within the more linear In Cycle (IC) phase. Furthermore, the paper describes a metamodel for modeling these so-called SCMs based on proven MBSE principles [7]. Since the models are being defined outside of an aircraft program when requirements are not yet fixed, the models have to be parametric. An SCM is a self-contained model that can be developed OOC and enables capturing of all information relevant to the development of the component. SCMs are foreseen to be stored in a repository, called the *SCM Library*. This enables sharing and reuse. When the IC phase of an aircraft or aircraft system development starts, the assets in the *SCM Library* are pulled and used as predefined and pre-verified components for a new development. The SCM metamodel defines all objects and their relations that are required to capture information related to SCMs. The development of the SCM metamodel was driven by internal use cases and inspired by existing modeling languages such as the Systems Modeling Language (SysML) [8].

The requirements for the methodology supporting this new OOC process were as follows:

- The methodology shall be based on MBSE principles.

- The methodology shall be independent from any specific application domain.
- The methodology shall enable a product-line oriented product development, i.e., the metamodel must allow modeling of different variants of a product and ensure a consistent configuration and parametrization.
- The methodology shall enable inclusion of already existing domain models, i.e., models in a domain-specific modeling language.
- The methodology shall enable automatic verification of models, i.e., it shall be possible to check if the built models adhere to the modeling paradigm and to user-defined constraints.
- The methodology shall enable consistent modeling not only of the product itself but also of the context, such as the industrial system used to build the product and allow the creation of relationships between the modeled artifacts.

The requirements for the application framework supporting this new modeling paradigm are as follows:

- The application framework shall be deployable in the current corporate IT infrastructure
- The application framework shall allow a heterogeneous technology stack to deliver the best solution for a designated purpose.
- The application framework shall be scalable with increasing number of models and users.
- The application framework shall be scalable in terms of model calculation performance.
- The application framework shall support continuous deployment strategies and agile frameworks to enable fast delivery and high flexibility.
- The application framework shall be efficient with regards to computing resources and reduce the company's ecological footprint.

A. OOC Process Description

The system lifecycle process as applied by most modern transportation system manufacturers including Airbus includes Design, Development, Production, Operation, Support, and Disposal (Figure 1).

Figure 1. System lifecycle according to [9]

As described in [9], according to the systems engineering approach, the system design process can be further divided into four major phases: conceptual design phase, preliminary design phase, detailed design phase, and test and evaluation phase (Figure 2).

The starting point for the current design process are requirements. Based on these requirements, initial design concepts are elaborated, assessed according to their feasibility, and evaluated against key performance indicators such as operating cost, weight, and range. A few concepts are then selected and refined in a preliminary design phase, and after a more profound analysis one of the design alternatives is chosen

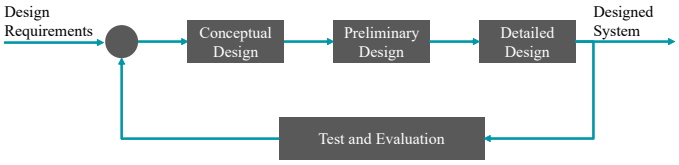


Figure 2. Major design Activities according to [9]

and further refined during detailed design analysis. To support the assessment of design concepts, various models describing different aspects of the aircraft system are developed. However, developing models to describe design alternatives is usually expensive and time consuming. In practice, when a new aircraft program is launched time pressure tends to lead to a situation where only very few alternative design concepts can be defined and assessed.

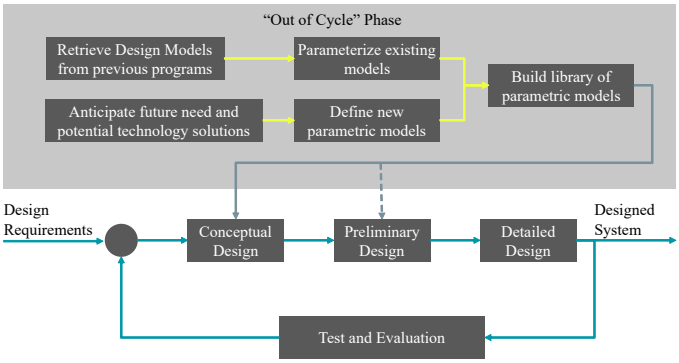


Figure 3. To-Be process with OOC phase

To address this challenge, it is suggested to introduce a new OOC phase as depicted in Figure 3. This phase is independent from aircraft programs and theoretically never ends. The aim is to produce reference architectures. Reference architectures will be decomposed into SCMs which simultaneously embed the knowledge of product design, its manufacturing system, operability, maintainability, cost and lead time. A SCM describes the technical solution for an architecture item in a reference architecture decomposition and its interfaces to other parts of the architecture, i.e., to other SCMs. Within the OOC phase, a library of SCMs of aircraft systems and components shall be defined that will grow over time. When a new aircraft program is launched, these models can be used to set-up different design concept alternatives. This shall save time and allow definition and analysis of a greater number of design alternatives, including more radical design concepts.

As shown in Figure 4, the OOC process can have different phases, during which the SCMs evolve and are being refined. The general idea is that a component matures in the OOC process until it is potentially ready to be used in an actual aircraft program. Once this stage is reached, the SCM is uploaded to a central library. At any time during the development of a new aircraft, the program can decide to pull the generic and parametrized component out of the library and specialize it to its needs. This process increases the reuse of the SCMs across multiple different programs resulting in an overall cost reduction and a decrease in the time to market for new products. This allows Airbus to react more quickly to the

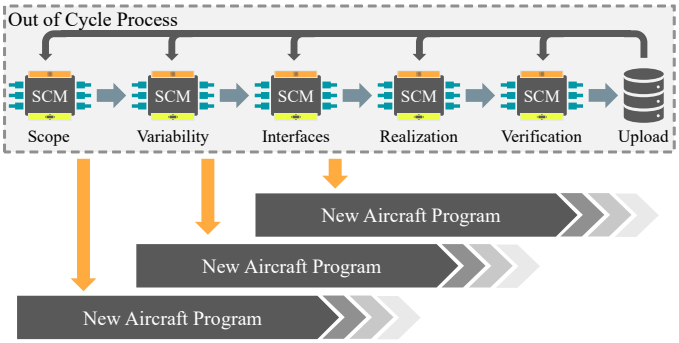


Figure 4. Evolution of a SCM over time

ever changing demands of the aerospace market.

Current design artifacts, however, do not have the features required to deal with not-yet specified product configurations and to support product evolution. They are not able to anticipate all features at design time.

Since the SCMs are being defined outside of any particular aircraft program, when requirements are not yet fixed, they have to be parametric in order to anticipate scalability and variability features and enriched with their associated limits. In case that the models are originating from previous programs, technologies have to be applied to parametrize them. Alternatively, the models may also be defined from scratch in a parametric way based on anticipation of future needs and technology trends.

SCMs provide an opportunity to capitalize interconnected multi-functional knowledge present in the organization, and also the capability to quickly generate new product designs by efficiently generating parametrized versions of components while maintaining its consistency in the overall architecture and considering its impact on integration and manufacturing processes. Their use will also naturally encourage reuse practices, which promise to make the development cycle faster and more cost-efficient.

B. Relation to other modeling languages

The most popular general purpose modeling language for systems engineering is SysML [8], which is itself an adaption of Unified Modeling Language (UML) [10] for systems engineering, yet it has still not become widely accepted [11], [12]. Karban et al. state challenges in using SysML, which have been figured out in the Active Phasing Experiment (APE) project of the SE2 challenge team of the Gesellschaft für Systems Engineering, German INCOSE chapter (GfSE) [13]. They propose several tasks for the advancement of SysML, which underlines that the language is still under development and will be further advanced in the future. Common points of criticism are: that SysML is too complex, which in turn causes complexity of SysML modeling tools; that it lacks a precise semantic; and that it does not come with a ready to use methodology, which is rooted in the fact that SysML was designed as a general purpose modeling language that should not impose a certain modeling approach. Currently, a completely reworked version 2 of SysML is being developed with the goal to increase adoption and effectiveness of MBSE by enhancing precision and expressiveness of the language, consistency and integration among language concepts and usability by model developers and consumers.

On the other hand, there are Domain-specific language (DSL), languages that are tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general-purpose modeling languages in their domain of application but generally require both domain knowledge and language development expertise to develop [14].

It seems to be an interesting question whether it is better to develop a new DSL from scratch by adding modeling elements iteratively, or start with a general purpose modeling language and restricting it until it fits the specific use.

[15] is talking about a ”yo–yo effect here: in the 1990s, many methods and modeling languages were popularized. [15] is talking about a ”yo–yo effect here: in the 1990s, many methods and modeling languages were popularized. Then, for a while, unification based on UML was very helpful. T hen, DSLs that were developed from scratch began to emerge. The next trend may be a repository of UML/SysML-based DSLs that actually unify DSLs and UML/SysML thinking.”

Our approach can be considered such a unified thinking. As already explained, we define our own DSL but it is closely aligned with SysML and we try to diverge only when we see a possibility for improving beyond the standard.

III. Architecture Paradigms

This Section provides background information regarding the two main architecture paradigms that are used today: monolithic software and MSA. Service-oriented architectures (SoA) and serverless architecture [16] are not described in detail as SoA, especially from a deployment perspective, still resembles monolith software [17] and serverless can be seen as taking MSA one step further [18].

A. Monolithic software

[19] defines a monolith as ”a software application whose modules cannot be executed independently”. This architecture is a traditional solution for building applications. A number of problems associated with monolithic applications can be identified:

- Due to their inherent complexity, they are hard to maintain and evolve. Inner dependencies make it hard to update parts of the application without disrupting other parts.
- The components are not independently executable and the application can only be deployed, started and stopped as a whole [20].
- They enforce a technology lock-in, as the same language and framework has to be used for the whole application.
- They prevent efficient scaling as popular and non-popular services of the application can only be scaled together [21].

Nevertheless, monolithic software is still widely used and, except for green-field new developments, there is hardly a way around it. [22] notes that a monolithic architecture is ”often a more practical and faster way to start”. Furthermore, if software from external parties is involved in a tool chain, it is not possible to change its architecture style.

B. Microservices

There is no single definition of what a MSA actually is. A commonly used definition by Lewis and Fowler says it is ”an approach for developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an Hypertext Transfer Protocol (HTTP) resource Application Programming Interface (API)” [23]. Microservices typically consist of stateless, small, loosely coupled and isolated processes in a ”share-as-little-as-possible architecture pattern” [24] where data is ”decentralised and distributed between the constituent microservices” [25].

The term ”microservices” was first introduced in 2011 [23] and publications on architecting microservices are rapidly increasing since 2015 [26]. In 2016, a systematic mapping study found that ”no larger-scale empirical evaluations exist” [27] and concluded that MSA is still an immature concept.

The following main benefits can be attributed to MSA:

- Relatively small components are easier for a developer to understand and enable designing, developing, testing and releasing with great agility.
- Infrastructure automation allows to reduce the manual effort involved in building, deploying and operating microservices, thus enabling continuous delivery [26].
- It is less likely for an application to have a single point of failure because functionality is dispersed across multiple services [17].
- MSA does not require a long-term commitment to any single technology or stack.

[4] notes the obvious drawback of the current popularity of microservices that ”they’re more likely to be used in situations, in which the costs far outweigh the benefits” even when monolithic architecture would be more appropriate.

In a study regarding the challenges of adopting microservices, [3] lists the distributed nature of MSA, which leads to debugging problems, the unavailability of skilled developers with intimate knowledge of MSA and finding an appropriate separation into services.

IV. Deployment Infrastructure

Corporate IT environments imply very strict regularities when it comes to hard- and software architectures and deployments. Bringing in innovation in such an environment requires following a heterogeneous approach.

While it is more challenging to adapt hardware in a corporate context to cope with the latest innovations, service and software developments, e.g., Advanced RISC Machine (ARM) Central Processing Unit (CPU) platform based servers, Graphics Processing Unit (GPU) assisted computing or wide-usage of Field Programmable Gate Arrays (FPGAs), the application platform layer adaption is typically less demanding because almost any state-of-the-art deployment form, like bare-metal, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or PaaS can be rolled out on standard server hardware.

The rationale for choosing a specific deployment form is based on various constraints imposed by corporate policies and long-term strategy decisions:

- Is the envisaged deployment form available in the corporate infrastructure?
- Has the deployment form limitations due to corporate policies, e.g., restricted internet access, restricted repository access?
- Are there any license limitations?
- Are there geolocation limitations for certain services, e.g., in a multinational company with multinational regulations according to law?
- Is the service available on premise or only on public cloud?
- Does a deployment form for a particular service fit in the long-term corporate IT strategy, e.g., make or buy decisions?

For the SCM modeling prototype, it was necessary to make use of a heterogeneous software and hardware infrastructure provided by the corporate IT. Therefore, the deployment took place on IaaS, PaaS and Function as a Service (FaaS) platforms. Also, end user devices are involved, for example for running the *SCM workbench* (see Figure 14). That variety of platform types was chosen to provide inside information on how a new engineering concept could be supported by different software architecture approaches to be efficient in terms of development time, Continuous Integration (CI), resource efficiency and scalability.

A. Infrastructure as a Service

In the context described above, IaaS is used to describe a hosting platform based on bare-metal and hosted hypervisors. It provides a variety of virtualized operating systems that are in compliance with corporate IT regulations.

For the prototype, the services hosted on classical virtual machines are mainly databases used as persistent layers for distributed Web applications. The main reason for not hosting the web applications together with their respective persistence layer are resource restrictions. Current company policies prevent external access to the databases if they are part of the same microservice image as the hosting environment. This would either limit database management to a web-based command line interface or require the implementation of a Web service deployed in the same container. Also, other external services could not be used to access the databases. This limitation is purely based on a decision made by the company’s IT governance, but reflects day to day reality in corporate environments.

For any other Web application around the SCM prototype development, IaaS was avoided as the resource overhead cannot compete with PaaS or FaaS.

B. Platform as a Service

In the following Section, PaaS refers to an on-premise deployment of the *Red Hat OpenShift* [28] platform. It is a platform built around *Docker* [29] containers orchestrated and managed by *Kubernetes* on a foundation of *Red Hat Enterprise Linux*.

In the prototype, PaaS plays a critical role for the continuous integration strategy. The image format used for the deployments follows the Source-to-image (S2I) concept. S2I is a toolkit and workflow for building reproducible container images from source code [30]. S2I produces ready-to-run images by injecting source code into a container image and

letting the container prepare that source code for execution. The source code itself is hosted on an on-premise Github Enterprise [31] instance and the dependent resources are provided via an on-premise *Artifactory* [32] deployment that reflects the official sources of the required development environment such as Maven [33], npm, Python or NuGet.

The whole continuous deployment chain is secured via an exchange of keys and certificates to prevent disruptions for example due to company introduced password cycles for the developer and deployment accounts. The deployment speed is improved by using system instances for the S2I chain in the same geolocation of the company to prevent larger inter-site data transfers and round-trip times.

The microservice concept, together with PaaS, allows a massive reduction of resource allocations compared to an IaaS deployment, especially if the services are single and independent web applications.

There are still limitations in the corporate environment that currently prevent larger scale use of the technology. The current setup allows a limited number of pods per node, which becomes an issue when a service uses the scaling capability of the *OpenShift* platform. A second limitation is linked to the allocated sub-network and the deployment of the platform. All inter-service communication is routed via a unique company internal network. The PaaS instance does not re-use a network range that is already present in the company for inter-service communications as it would impose other challenges regarding communication from within the PaaS instance towards other company services. The rationale for the chosen PaaS implementation is primarily the reduction of classical virtual machines for simple hosting jobs and only secondarily the creation of a massively scalable infrastructure for new service applications.

To cope with these limitations the prototype furthermore reduces the deployment footprint of single services for certain applications as described below.

C. Function as a Service

FaaS is used for tiny stateless jobs, e.g., rendering of images. These services are monitored by an orchestrator that decommissions containers after idling for a defined time. This reduces resource usage further and has advantages in a scenario with a larger number of services.

The deployment architecture of the FaaS instance allows launching service containers within milliseconds. The applied software stack is *OpenFaaS* based on *Docker Swarm* running on a *Debian* [34] Virtual Machine (VM).

One FaaS instance consumes resources similar to a pod on the above mentioned PaaS environment and hosts numerous services without performance limitations. While PaaS exposes containers under their distinct Internet Protocol (IP) addresses, FaaS comes with a reverse proxy that hides all containers and requires less IP addresses. This reduces the effort for routing name resolution and their documentation.

V. Implementation and Integration

The implementation of the prototype framework is split into different logical bricks as depicted by Figure 5. The services and applications itself can be mapped to their specific deployment paradigm as listed in Table I. The *Architect Cockpit* allows a system architect to use existing models, to schedule the execution of simulations and to review results. The *SCM Workbench* enables SCM developers to create and version SCMs. The *Back End* provides different services such

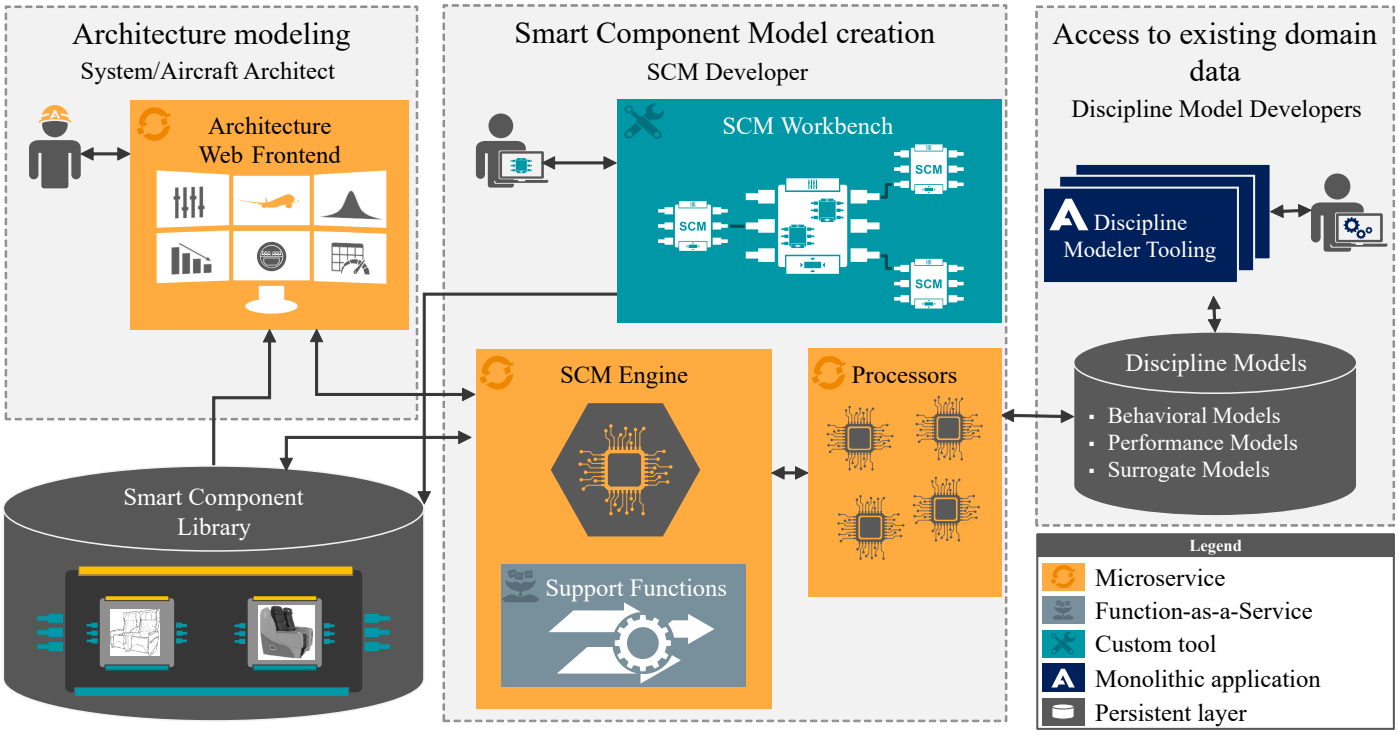


Figure 5. Service Environment & Deployment Infrastructure

as the orchestration of different processors to perform the execution of simulations.

TABLE I. Service Mapping to Deployment Paradigm

Deployment Paradigm	Service or Application
IaaS	Persistence Layers (Smart Component Library, MongoDB, Internal GitHub, Internal Artifactory)
PaaS	SCM Engine, Architecture Web Frontend, Service Dashboard, OpenTURNS Sampler, Node-Red, SCM Processors, Jenkins
FaaS	JSONata, Parallel-Coordinates
End User Device	SCM Workbench

A. *Architect Cockpit*

In order to reduce the workload and make the work for the architects as convenient as possible the interface for the cockpit is setup as an Angular Single Page Application (SPA). This allows using this entity without installing custom software and without bothering the user with update and migration procedures. The site is built using a Jenkins pipeline and then deployed on a specific git repository branch. A webhook on this branch triggers an *OpenShift* instance to build an Express.js server serving the previously build site on a PaaS cluster.

From a functional point of view the *Architect Cockpit* gives a reduced view on SCMs. Only information, which is necessary for the work of an architect is available and can be modified. This results in a nearly full intuitive usage of the interface and prevents faulty configurations. For example, some parameters can only be changed within a certain range. Ranges are defined by the model developer who knows the limitations best. The architect does not need to have a deep understanding of these limitations when using the predefined models.

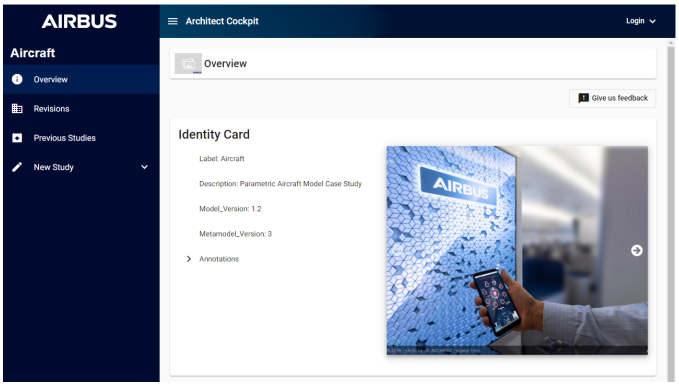


Figure 6. Architect entry point for gathering information on an SCM

Figure 6 shows the Architects entry point into using a SCM. In this case it is an aircraft architect opening a parametric aircraft model. He can then start a new study and set or change parameters of this model which Figure 7 shows. After running a calculation in the mixed-paradigm back-end the front-end renders an executive result and presents it to the architect. Figure 8 shows the result. If deeper insights into the calculation chain is required the architect can open a more

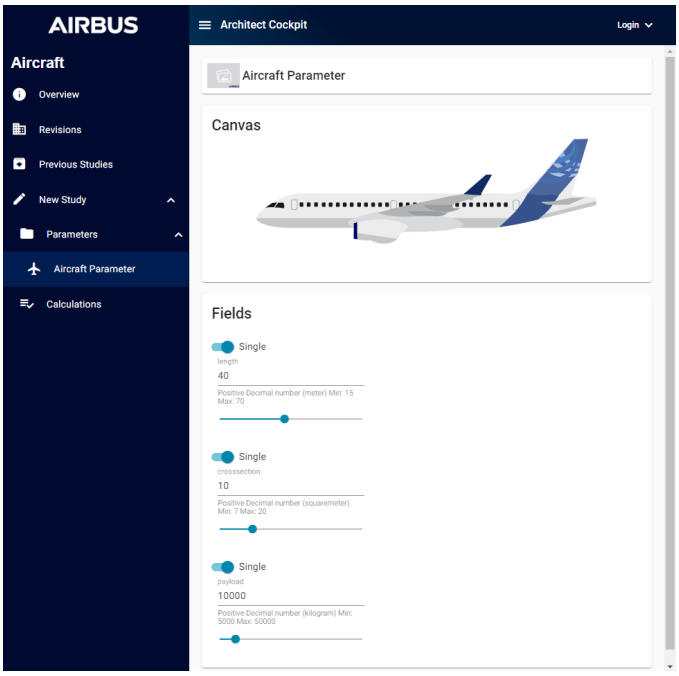


Figure 7. User interface for defining parameter values

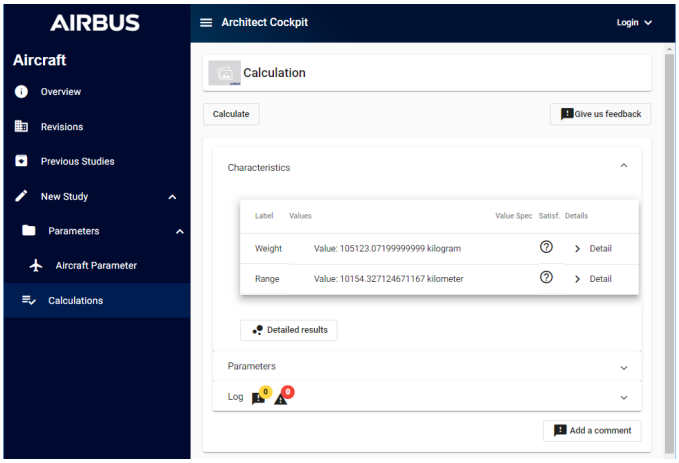


Figure 8. User interface for launching calculation with a SCM

detailed interactive report rendered as a bubble chart as shown in Figure 9. When selecting a range of values for a set of parameters an additional representation appears rendering all distinct runs of a study into a parallel coordinates plot. Figure 10 shows this interactive diagram. It is a data analytics tool that allows highlighting specific runs, filtering specific parameter and characteristic values as well as removing parameters from the diagram.

B. *SCM Workbench*

The *SCM Workbench* is a full-fledged graphical editor to work with SCMs implemented as a monolithic rich-client application. It is implemented in an Eclipse Rich Client Platform (RCP) and based on the Eclipse Modeling Framework (EMF) [35]. It is a modeling framework and code generation facility for building tools and other applications based on a

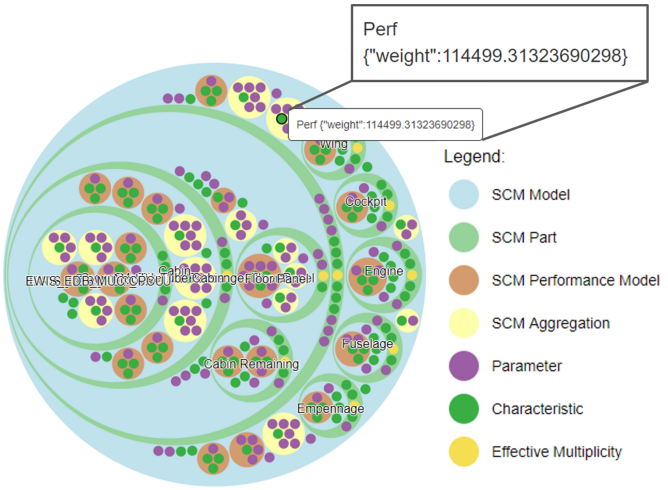


Figure 9. Bubble chart for browsing through the propagation of calculated values

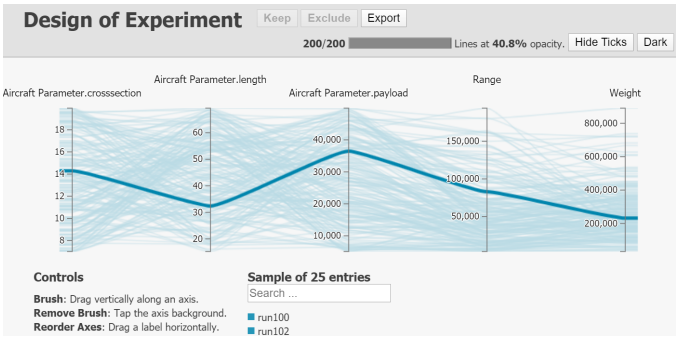


Figure 10. Design of Experiment explorer for navigation through large sets of sampled simulation runs

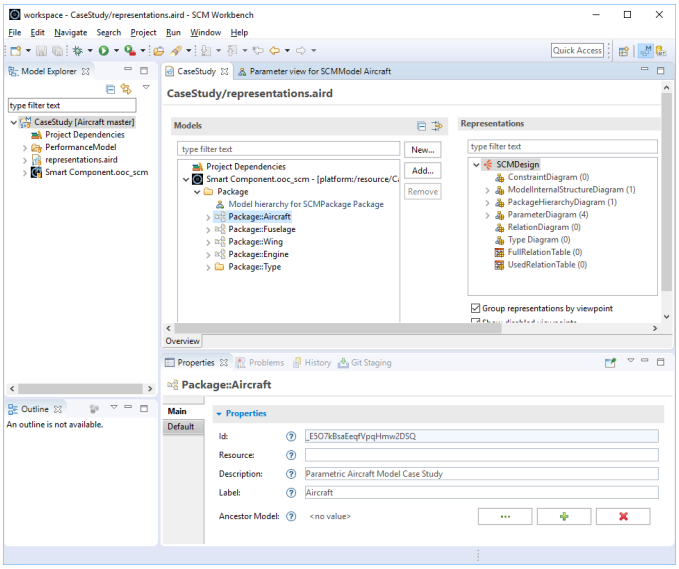


Figure 11. SCM Workbench showing the representations

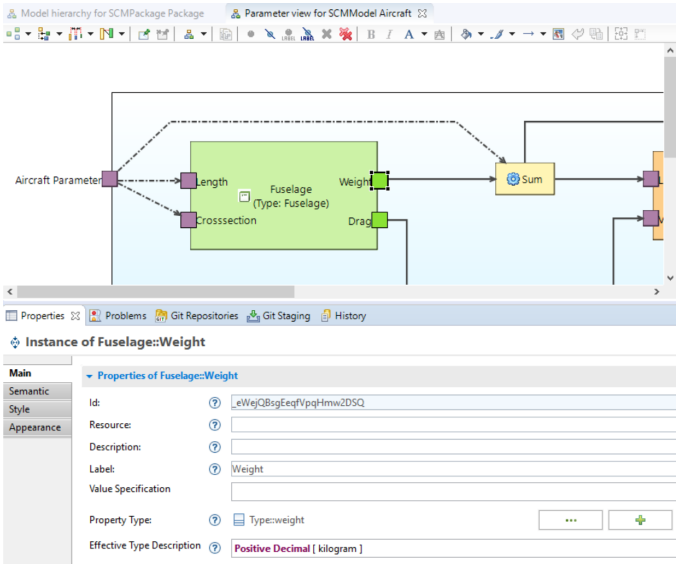


Figure 12. Parameter view in SCM Workbench

structured data model. EMF provides tools and run-time support to produce a set of Java classes from a model specification, along with a set of adapter classes that enable viewing and editing of the model, and a basic editor.

EMF is the basis for the Obeo Designer tool [36], which builds on the Eclipse Sirius project [37] and allows definition of graphic editors based on a defined EMF metamodel. This enables rapid prototyping of modeling solutions, which is ideal for a research/prototyping environment such as Airbus Central R&T. Changes to the metamodel are almost instantly available in the *SCM Workbench*, our prototype SCM modeling tool. On the other hand, EMF and *Obeo Designer* are mature and have been proven in industrial practice, e.g., *Capella*, the modeling tool from Thales that implements the *Arcadia* method is built with EMF and *Obeo Designer* as well [38].

Starting the *SCM Workbench* opens an application that is shown in Figure 11. The left toolbar allows browsing through the SCMs that exist in a project. A project is split into SCMs and representations. While the SCMs contain all functionality required for computing it, the representation adds additional information on how to render the SCMs within the workbench. Figure 12 shows the parameter view that allows the *SCM Developer* to model the propagation of parameters and characteristics through the SCM. Selecting entities in this view leads to its properties to show up in an editor at the bottom of the window. This example shows the parameter ”Weight” of the SCM ”Wing”. The structure view shown in Figure 13 describes the architectural interdependency between underlying SCMs. In this case the ”Engine” is attached at the ”Pylon” to the ”Wing” and the ”Wing” is attached to the ”Fuselage” at the ”Belly”.

Using such a rapid prototyping approach for the *SCM Workbench* can be easily misunderstood as just a proof-of-concept study. The final look and feel of the graphical editor for the SCMs is only limited by the amount of development time used for user experience (UX) polishing. The workflow and information accessibility as well as the connection to a versioning system is comparable to other commercially available modeling tools, which are well known by the developers. It is assumed that a SCM developer has to take a short on-boarding training before using the *SCM Workbench*.

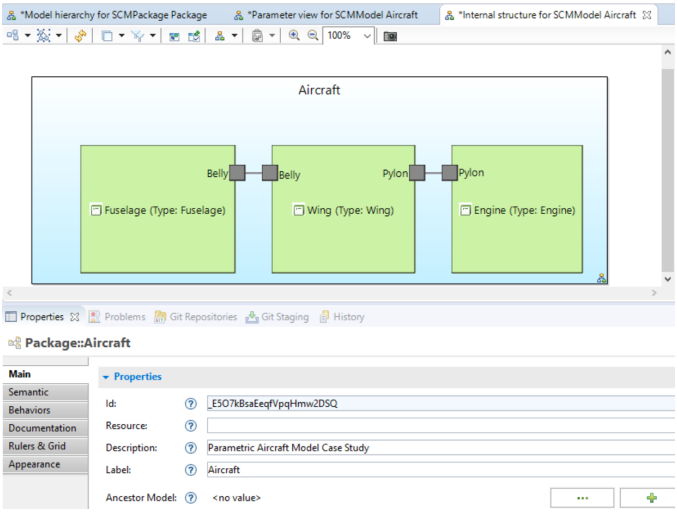


Figure 13. Structure view in SCM Workbench

C. Back End

The *back-end* is built from several different entities that are based on different paradigms. These entities are described in the following paragraphs.

1) SCM Library: The *SCM Library* stores the models that have been created using the *SCM Workbench*. It is based on Connected Data Objects (CDO) a Java model repository for EMF models and metamodels. The specific implementation in use is the *Obeo Designer Team Server (ODTS)*, which enables concurrent engineering of EMF models. A custom plug-in allows other services and applications to access the model repository through a Representational State Transfer (REST) interface. Due to its complex deployment strategy the *SCM Library* is deployed in an IaaS environment, which allows more user interaction during updates.

2) SCM Engine: The *SCM Engine* can interpret SCMs, check constraints and run parametric calculations either as a single simulation run or as a Design of Experiments (DoE) setup with multiple samples. It is a Java application executed in an OpenJDK VM. Access to the engine is established through REST interfaces that are hosted on a Jetty server. The endpoints are described and documented using the Jersey framework. The *SCM Engine* is hosted on a PaaS instance and allows rolling updates, automated builds and scaling.

3) Model Processors: The *Performance Model API* serves as a glue between external domain-specific models with their own solver or simulation engine and the *SCM Engine*. A *Model Processor* is an application that implements this API to execute a specific model type. The API enables the *SCM Engine* to orchestrate simulations tools in a unified way and guides developers through the process of integrating additional simulation tools into this environment. In order to include a new model type in the SCM application framework, a model type specific *Model Processor* has to be implemented that implements the *Performance Model API* and connects to the model type specific solver or simulator. A reference implementation shows how this works for Excel models. An Excel model is processed by a Java application running in an OpenJDK VM using the Apache POI framework. Depending on the type of model and, e.g., the license and installation requirements of the model solver or simulator, the *Model Processor* can be deployed in any of the available deployment options IaaS, PaaS and FaaS.

Figure 14 depicts how the components of the SCM tool framework prototype are deployed in our infrastructure.

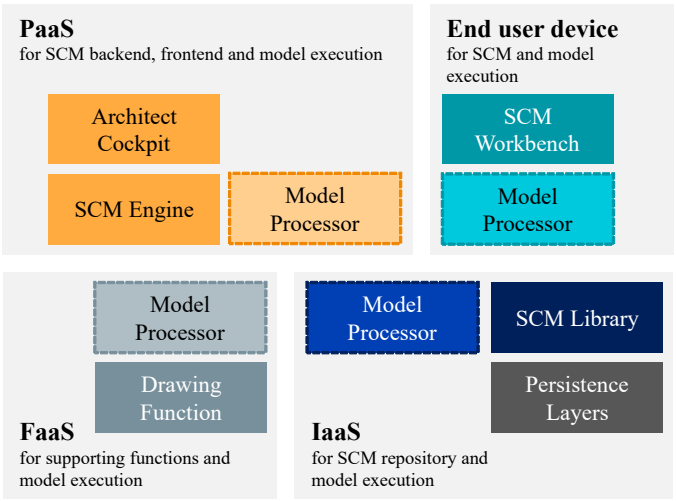


Figure 14. Prototype tool deployment

To make the polyglot approach of the MSA work and integrate each service all participating entities need to agree to a commonly understood interface. For the prototype REST over HTTP was chosen as the default interface combined with JavaScript Object Notation (JSON) as serialization format. REST over HTTP is a de facto standard since almost every technology stack provides at least an HTTP API if not specialized REST frameworks and clients such as Java API for RESTful Web Services (JAX-RS). JSON as a serialization format is accepted and provides solid tooling on all integrated technologies. In addition many front-end frameworks natively support JSON such as *JavaScript* or *Ruby*. This eases the integration work needed to be done for the implementation of our demonstrators mainly the *Architect Cockpit*. As an added bonus it is easily digestible by human user, which helped tremendously with debugging. To build up process chains utilizing the deployed microservices we selected *Node-RED*. It provides all the tools necessary to handle HTTP based REST APIs and JSON based message bodies and is integrated well into the existing environment.

If we dissect the service environment infrastructure shown in Figure 5 we can see what protocols are being used in the communication between the different services. As Figure 15 shows the communication through the HTTP RESTful web services is the predominate form of communication in our prototype. The only deviation from this paradigm occurs in the communication between the *SCM Workbench* and the *SCM Library* where the tool provider specifies Transmission Control Protocol (TCP) as interface. We did not challenge this implementation since it is provided by the *Obeo Designer Team Server*; however, we implemented a *Mapper Plugin* that provides access to the stored SCM Models via a REST interface to incorporate it in our service environment. The HTTP REST approach is especially useful for incorporating the various Discipline Models Processors into the overall pro-

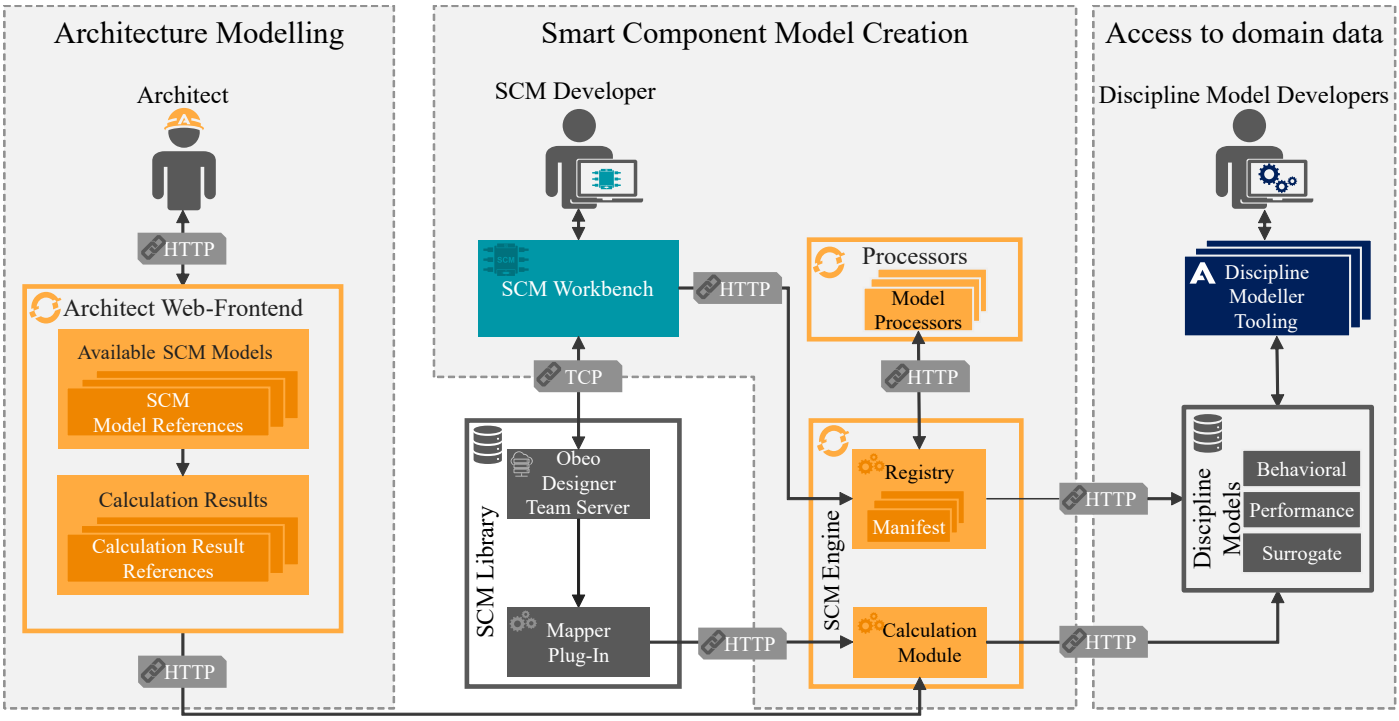


Figure 15. The use of REST interfaces in our prototype

cess. Since every processor works on a dedicated technology stack designed for his task utilizing a unified interface makes integration into the overall process network easy.

After the explanation of all the building blocks we will present a case study to demonstrate the framework in action.

VI. Case Study

This section describes a case study that has been made using the SCM Workbench with the purpose of showing individual features of the tool-chain.

The described model has been provided showing an Aircraft consisting of Fuselage, Wing and Engine. The hierarchical view, depicted in Figure 16, of the SCMs shows that all models are placed in one package. The hierarchical view is equivalent to SysML's class diagram. This view shows that the Aircraft is decomposed into Fuselage, Wing and Engine. However, it does not show their interdependency.

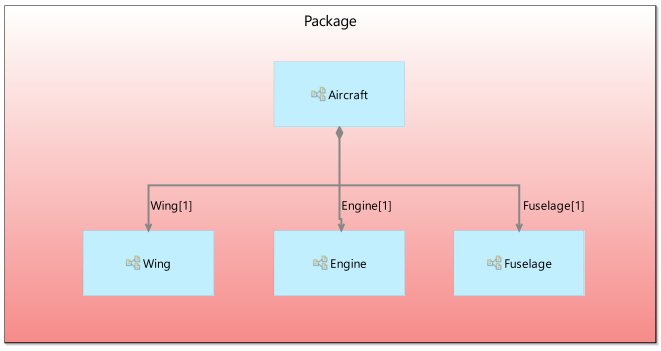


Figure 16. Case study: Hierarchy view

All models are SCMs. The Aircraft is a special case, because it is decomposed into other SCMs.

In order to describe the interdependency within the Aircraft SCM the internal structure diagram was used, shown in Figure 17. It shows that the Fuselage and Wing are connected with each other at the Belly. Wing and Engine are connected at the Pylon. This diagram is similar to SysML's internal block diagram. The blue box shows the scope of the SCM. The green boxes represent the decomposition into other SCMs.

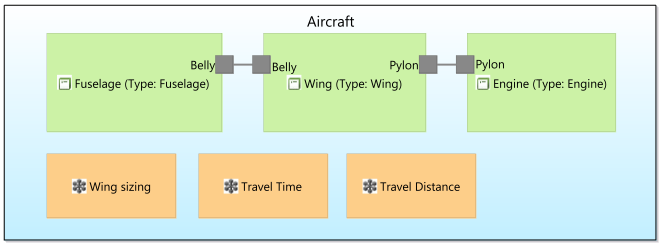


Figure 17. Case study: Structure view

Additionally there is the parameter view, shown in Figure 18, which describes in-transient calculation of the model characteristics from the set of parameters. All parameters appear in purple color and all characteristics in green color. This specific SCM describes the calculation of the Aircraft's weight and range from its length, cross-section and payload. Light yellow color represents an aggregation function that generates one value from a list of values. In this case both aggregation nodes compute a sum. Direct calculations on performance models are represented by the orange boxes. They refer to a domain model manifest, describe machine-readable how these domain models

shall be executed and how parameters and characteristics are transmitted to and from them.

In order to ensure compatibility between parameters and/or characteristics a common type system is available. It allows to specify the data to be exchanged during the simulation between the components. Besides the typical basic types it allows structured types like lists and key-value-pairs. An additional feature is to nest types as references into other types. As an example the Aircraft parameters are shown in Figure 19. As an example for a basic type the weight is shown in Figure 20.

VII. Evaluation

Evaluating the mixed-paradigm approach, we experienced that developers where able to create a working deployment much faster compared to the traditional approach using virtual machines. This also includes the amount of times that a new version of the service was built from once a week to several times a day using the automated CI pipeline. This increased the general development velocity as well as the prototypes feature set, which helped us to tailor the application to our stakeholder needs.

The raised deployment speed increased the number of times we experienced broken client applications. This was due to a violated interface contract between the services if the new features where not integrated properly. A well-defined and adhered to interface specification is paramount for the success of introducing this mixed-paradigm approach.

In general, we noticed a greater sense of ownership of single developers over their service/code, which lead to a hike in the overall implementation quality. The mandatory usage of the git version control system increased the maintainability of the code base. The combination of git and the *OpenShift* framework made it easy to recover from failures and faulty builds, which lead to a constant up-time of all services. In the future the introduction of additional agile software development principles like *Test Driven Development* could further increase the code quality.

However, the deployed solution is marked as a proof of concept or prototype which lead to the conclusion that it is not ready for operational use for various reasons. The main focus of the development lay on the proof of feasibility of the SCM modeling methodology as described in [6]. As soon as first parts of the prototype were available selected engineering departments started trials with the solution, which lead to further improvements of the underlying methodology as well as the overall usability. The overall perception was very positive, which lead to the conclusion that the developed methodology points in the right direction as well as the performance of the proposed tool set based on the described approach in this paper.

The mixed-paradigm approach that was used to develop and deploy the prototype discussed in this paper led to reduced complexity, lower coupling, higher cohesion and a simplified integration. This in turn enabled agile collaboration for continuous delivery and integration of the solution.

VIII. Outlook and future challenges

In the previous sections, we described how MSA can support the chosen polyglot approach utilizing a variety of different technology stacks and storage solutions. This enabled us to select the most fitting technical solution for the required functionality. Additionally the network based architecture provides an environment that is well suited for a multinational company like Airbus with sites scattered throughout different sites and IT domains. It also provided a commonly understood deployment layer for our cross-functional project team.

MSA supports us with the agility and velocity needed to convince our customers of our approach and implement a prototype that can handle the complexity of our SCM modeling approach. However, during the development we found stumbling blocks that need awareness once the scale changes from a research project prototype to a full scale industrial roll out.

Corporate IT – The proposed environment builds and hosts microservices in an agile and automated way. This requires the setup and maintenance of a CI pipeline (in our case *OpenShift*/*GitHub*), which results in additional costs as well as an IT department that is capable of dealing with those investments. Additionally setting up certificate chains and firewalls to allow for secure communication inside the corporate network need to be accounted for. On the developer side roadblocks like proxy server hindering communication and enabling cross-origin resource sharing (CORS), which allows for communication between different domains need to be taken care of.

Service discovery – Once we reached a critical mass of microservices environment we discovered that it is hard to keep track of what services have already been implemented and what functionality each service provides. Even in our research project this point was reached rather quickly. Thus, we introduced *Swagger* [39] as a Web based documentation for all our services and implemented a simple dashboard where services could be registered against. This allowed for manual service discovery across the team. In the future automated service discovery through bots and processable service descriptions will bring more value to the MSA approach by handling the sprawling service environment.

Now that we optimized the CI pipeline in the first half of the project we experience a rapid increase in deployed services. This allowed us to swiftly introduce new functionality as microservices, boosting the capabilities of our proof of concept prototype. It shows that MSA can initially speed up the implementation velocity of a new project. Once we continue with the project more efforts will go towards managing the volume of services as well as (network) performance and reliability.

IX. Conclusion

Past experience shows that current aircraft and aircraft system development processes are not suitable for keeping up with the rising complexity of products. Those processes are under pressure from market-driven demands for faster, and from business-driven demands for cheaper aircraft programs. In this paper, we presented a proposal for a change from a traditionally linear development approach to one that includes a parallel, OOC component development phase. This approach required a supporting IT infrastructure that was built as a prototype at Airbus in the frame of a research project. To reduce both time and resources required for building this prototype state-of-the-art architecture and deployment paradigms were

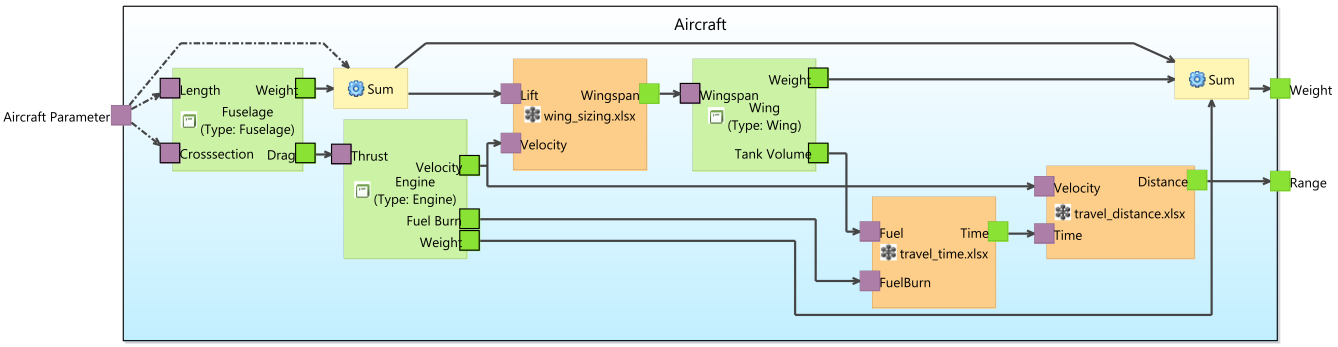


Figure 18. Case study: Parameter view

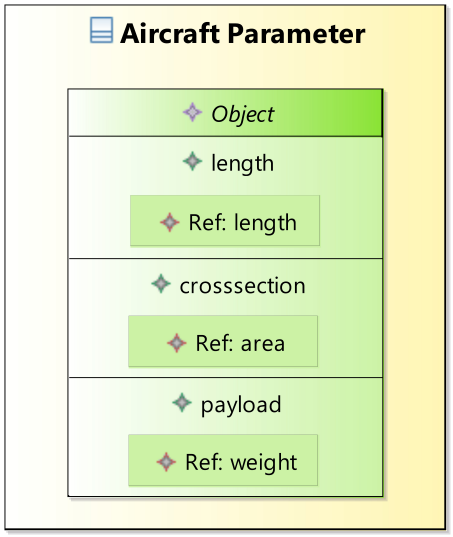


Figure 19. Case Study: Aircraft parameter modeled in the type system

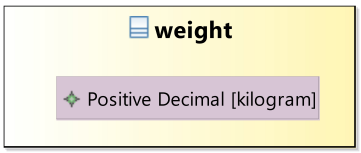


Figure 20. Case Study: Basic parameter representation in the type system

used and mixed with more classic approaches to get the best of both worlds.

A direct, specific and measurable comparison between the described mixed-paradigm and a classical approach is not possible as it would have required the same infrastructure landscape to have been developed and deployed multiple times using different concepts. Nevertheless, implementers were given the freedom to decide for every distinct artifact to freely choose the paradigm used for implementation. Furthermore, developers were allowed to split artifacts, which enables to select the right paradigm for each problem within. Later the interface documentation allowed the developers to easily re-implement an artifact using a different paradigm in case the initial decision for a specific paradigm reveals to have been not an optimal choice. Therefore, the selection of the right paradigm appears to be inherent and native. To support a newly developed MBSE approach called SCM modeling, a supporting application framework prototype had to be developed. Instead of a single architecture and deployment paradigm, a mixed-paradigm approach was followed to take the advantages of the different options and to consider external constraints coming from the IT governance. The software bricks were implemented in monolithic, SoA, microservice and serverless architecture glued together by REST interfaces over HTTP. The deployment took place on desktop-PC, IaaS, PaaS and FaaS platforms. It provided insight into how a new engineering concept could be supported by different software architecture approaches to be efficient in terms of development time, continuous integration, resource efficiency and scalability.

References

[1] P. Helle, S. Richter, G. Schramm, and A. Zindel, “Mixed-Paradigm Framework for Model-Based Systems Engineering,” in FASSI 2019, The Fifth International Conference on Fundamentals and Advances in Software Systems Integration. IARIA, 2019, pp. 8–13.

[2] N. Dragoni et al., “Microservices: yesterday, today, and tomorrow,” in Present and ulterior software engineering. Springer, 2017, pp. 195–216.

[3] J. Ghofrani and D. Lübke, “Challenges of microservices architecture: A survey on the state of the practice.” in ZEUS, 2018, pp. 1–8.

[4] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, “Microservices: The journey so far and challenges ahead,” IEEE Software, vol. 35, no. 3, 2018, pp. 24–35.

[5] D. D. Walden, G. J. Roedler, K. Forsberg, R. D. Hamelin, and T. M. Shortell, Eds., Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 4th ed. Hoboken, NJ: Wiley, 2015.

[6] P. Helle, S. Feo-Arenis, A. Mitschke, and G. Schramm, “Smart component modeling for complex system development,” in Proceedings of the 10th Complex Systems Design & Management (CSD&M) conference, forthcoming.

[7] A. Reichwein and C. Paredis, “Overview of architecture frameworks and modeling languages for model-based systems engineering,” in Proc. ASME, 2011, pp. 1–9.

[8] Object Management Group, OMG Systems Modeling Language (OMG SysML), v1.2. OMG, Needham, MA, 2008.

[9] M. H. Sadraey, Aircraft design: A systems engineering approach. John Wiley & Sons, 2012.

[10] Object Management Group, OMG Unified Modeling Language (OMG UML), v2.3. OMG, Needham, MA, 2010.

[11] J. A. Estefan et al., “Survey of model-based systems engineering (mbse) methodologies,” Incose MBSE Focus Group, vol. 25, no. 8, 2007, pp. 1–12.

[12] A. Albers and C. Zingel, “Challenges of model-based systems engineering: A study towards unified term understanding and the state of usage of sysml,” in Smart Product Engineering. Springer, 2013, pp. 83–92.

[13] R. Karban, R. Hauber, and T. Weilkiens, “Mbse in telescope modeling,” INCOSE INSIGHT, vol. 12, no. 4, 2009, pp. 24–31.

[14] A. Van Deursen, P. Klint, and J. Visser, “Domain-specific languages: An annotated bibliography,” ACM Sigplan Notices, vol. 35, no. 6, 2000, pp. 26–36.

[15] J. Gray and B. Rumpe, “Uml customization versus domain-specific languages,” 2018.

[16] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, “Serverless programming (function as a service),” in 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2017, pp. 2658–2659.

[17] A. Karmel, R. Chandramouli, and M. Iorga, “Nist definition of microservices, application containers and system virtual machines,” National Institute of Standards and Technology, Tech. Rep., 2016.

[18] I. Baldini et al., “Serverless computing: Current trends and open problems,” in Research Advances in Cloud Computing. Springer, 2017, pp. 1–20.

[19] N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, “Microservices: Migration of a mission critical system,” arXiv preprint arXiv:1704.04173, 2017.

[20] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, “From monolithic to microservices: an experience report from the banking domain,” Ieee Software, vol. 35, no. 3, 2018, pp. 50–55.

[21] M. Villamizar et al., “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud,” in 2015 10th Computing Colombian Conference (10CCC). IEEE, 2015, pp. 583–590.

[22] ——, “Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures,” Service Oriented Computing and Applications, vol. 11, no. 2, 2017, pp. 233–247.

[23] M. Fowler and J. Lewis. Microservices a definition of this new architectural term. [Online] http://martinfowler.com/articles/microservices.html [Accessed: 11 September 2019].

[24] T. Cerny, M. J. Donahoo, and M. Trnka, “Contextual understanding of microservice architecture: current and future directions,” ACM SIGAPP Applied Computing Review, vol. 17, no. 4, 2018, pp. 29–45.

[25] D. Shadija, M. Rezai, and R. Hill, “Towards an understanding of microservices,” in 2017 23rd International Conference on Automation and Computing (ICAC). IEEE, 2017, pp. 1–6.

[26] P. Di Francesco, I. Malavolta, and P. Lago, “Research on architecting microservices: trends, focus, and potential for industrial adoption,” in 2017 IEEE International Conference on Software Architecture (ICSA). IEEE, 2017, pp. 21–30.

[27] C. Pahl and P. Jamshidi, “Microservices: A systematic mapping study.” in CLOSER (1), 2016, pp. 137–146.

[28] RedHat, “Openshift,” https://www.openshift.com/, 2019, [Online; accessed 3-February-2020].

[29] Docker Inc., “Docker,” https://www.docker.com/, 2019, [Online; accessed 3-February-2020].

[30] A. Lossent, A. R. Peon, and A. Wagner, “PaaS for web applications with OpenShift origin,” Journal of Physics: Conference Series, vol. 898, oct 2017, p. 082037.

[31] GitHub, Inc., “Github,” https://github.com/, 2019, [Online; accessed 3-February-2020].

[32] JFrog Ltd, “Artifactory,” https://jfrog.com/artifactory/, 2019, [Online; accessed 3-February-2020].

[33] The Apache Software Foundation, “Maven,” https://maven.apache.org/, 2019, [Online; accessed 3-February-2020].

[34] Software in the Public Interest, Inc., “Debian,” https://www.debian.org, 2019, [Online; accessed 3-February-2020].

[35] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, EMF: eclipse modeling framework. Pearson Education, 2008.

[36] Obeo, “Obeo designer,” https://www.obeo.fr/en/, 2019, [Online; accessed 3-February-2020].

[37] V. Viyović, M. Maksimović, and B. Perisić, “Sirius: A rapid development of dsm graphical editor,” in IEEE 18th International Conference on Intelligent Engineering Systems INES 2014. IEEE, 2014, pp. 233–238.

[38] P. Roques, “MBSE with the ARCADIA Method and the Capella Tool,” in 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Toulouse, France, Jan. 2016.

[39] SmartBear Software, “Swagger,” https://swagger.io/, 2019, [Online; accessed 3-February-2020].

Data Science as a Service

Prototyping an Integrated and Consolidated IT Infrastructure Combining Enterprise Self-Service Platform and Reproducible Research

Hans Laser
Center for Information Management
Hannover Medical School
Hannover, Germany
e-mail: laser.hans@mh-hannover.de

Steve Guhr
NetApp Deutschland GmbH
Berlin, Germany
e-mail: steve.guhr@netapp.com

Jan-Hendrik Martenson
NetApp Deutschland GmbH
Hamburg, Germany
e-mail: jan-hendrik.martenson@netapp.com

Jannes Gless
Center for Information Management
Hannover Medical School
Hannover, Germany
e-mail: gless.jannes@mh-hannover.de

Branko Jandric
Center for Information Management
Hannover Medical School
Hannover, Germany
e-mail: jandric.branko@mh-hannover.de

Joshua Görner
Airbus Operations GmbH
Hamburg, Germany
e-mail: joshua.goerner@gmail.com

Detlef Amendt
Center for Information Management
Hannover Medical School
Hannover, Germany
e-mail: amendt.detlef@mh-hannover.de

Benjamin Schantze
NetApp Deutschland GmbH
Hamburg, Germany
e-mail: benjamin.schantze@netapp.com

Svetlana Gerbel
Center for Information Management
Hannover Medical School
Hannover, Germany
e-mail: gerbel.svetlana@mh-hannover.de

***Abstract*–A data scientific process (e.g., Obtain, Scrub, Explore, Model, and iNterpret (OSEMN)) usually consists of different steps and can be understood as an umbrella for the combination of different most modern techniques and tools for the extraction of information and knowledge. When developing a suitable IT infrastructure for a self-service platform in the academic environment, scientific requirements for reproducibility and comprehensibility as well as security aspects such as the availability of services and of data are to be taken into account. In this paper, we show a prototypical implementation for the efficient use of available data center resources as a self-service platform on enterprise technology to support data-driven research.**

Keywords-data science as a service; reproducible research; enterprise information technology; research data infrastructure; self-services; data science platform; cloud infrastructure.

I. INTRODUCTION

One of the most important aspects of building service portfolios in the company is to make them as simple and usable as possible for the end user [1].

Data centers at German universities are increasingly confronted with challenges in the areas of availability and operational security, data privacy and IT security, operating costs and use of cloud services, data management and access to high computing capacity, increasing standardization and consolidation of IT systems [2]. The availability of services and especially of data requires a corresponding infrastructure. Services are increasingly subjected to risk classification in order to define how the operation of university IT must be ensured. The technology used must allow to scale out (horizontally) performance of a platform, because scaling up a platform is limited to hardware resources. Today, virtualization technology is often used to

compress computing power and at the same time to have the flexibility to move parts of the computing nodes and redistribute the load in case of a failure. Hardware maintenance does not necessarily result in the unavailability of a service, as it can usually be moved without interruption to another node in the data center cluster or even the cloud [3]. This infrastructure is supplemented by container technology as a further step in service encapsulation. At the same time, similar demands are made on the availability of data. Redundant Arrays of Independent (Inexpensive) Disks (RAID systems) combine performance, reliability and scalability but show weaknesses in the speed of recovery [4]. Monitoring the health of IT infrastructures with suitable tools is indispensable for professional operation [5].

The main challenge is to "transfer" (integrate) these well-known tools and solutions into a scalable and powerful platform that can be equipped with enterprise technology to ensure service level agreements (SLAs) from a centralized enterprise information technology [3].

Users of documented services are provided with more specific minimum and maximum performance measures, such as quality, punctuality or cost of a service, through the SLAs and can adapt and understand the expectations and limitations of a service accordingly [6].

The process of combining or connecting different systems and individual software applications to form a common functional system with comprehensive functionality to increase user acceptance and customer satisfaction is also known as system integration. IT service providers have an interest in the continuous improvement of product and service quality [7]. The added value for the company can be increased by improving service response times, reducing the costs for the operation of IT infrastructure, and lowering operating costs by intelligently linking the IT systems used [8].

A. The Data Science Process

To obtain information (e.g., based on patterns) for relevant business decisions from data of heterogeneous data sources, a classical multi-stage process for data preparation and analysis is used, the so-called data mining process [9]. Data science, on the other hand, can be understood as an umbrella for the combination of various state-of-the-art techniques for the extraction of information and knowledge (so-called insights) to develop data-based applications and, thus, automating processes. One approach to describe the individual steps for the data science process is Obtain, Scrub, Explore, Model and INterprenting (OSEMN) [10]. In the Obtain step, for example, query languages are required for databases that can be extracted in various formats. Python [11] and R [12] encapsulate the otherwise heterogeneous data query tools (e.g., Structured Query Language (SQL), eXtensible Markup Language (XML), Application Programming Languages (API), Comma Separated Value (CSV), Hybrid File System (HFS)). Classic database techniques such as Extract Transform Load (ETL) process can be used in the cleanup step (Scrub). Computer languages like Python and R or application suits like SAS Enterprise Miner [13] or OpenRefine [14] can also be used to transform data. To examine the data (Explore) languages like Python or R specialize in particular appropriate libraries (e.g., Pandas [15] or Scipy [16]). In this step, however, familiar players from the business intelligence world (e.g., Rapid Miner [17] or KNIME [18]) can also be found for data-wrangling. To build a model, there are again specialized Python libraries like “Sci-kit learn” [19] or CARET [20] for R. Other tools like KNIME or Rapid Miner find reuse in this step as well. Finally, for interpreting the model and the data as well as evaluating the generalization of the algorithm, tools for data visualization are reused (e.g., matplotlib [21], Tableau [22] or MS Power BI [23]). In summary, it means, that for the many single steps in OSEMN, many different tools can be necessary.

An example of a platform solution that maps these process steps in a so-called pipelining functionality is the Pachyderm software [24]. Pachyderm offers components that support the developer (data scientist) with regard to data provenance in development work and analyses and can thus map a logical and chronological sequence of process steps. This platform solution offers many degrees of freedom and requires the user to have a well-developed hypothesis or data processing or data management plan and is therefore suitable for the development of concrete and declarable products [25]. However, if the workflow initially requires exploration for hypothesis generation, it may be better to use tools of lower complexity.

B. Reproducible Research

Scientific studies, experiments and numerical calculations can only be reproduced or reconstructed if all important steps are comprehensible [26]. The importance of reproducibility of scientific work can be illustrated by the following quotation from Jon Clearbout: “An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.” [27]

A scientist should therefore always have an interest in describing the runtime environment as transparently and understandably as possible. However, complex runtime environments are difficult to penetrate due to the sometimes high technological complexity (e.g., package dependencies). The technical reproducibility enables scientific results to be reproduced at all, but this requires a very high degree of knowledge about the method and technology used. Simplifying the technological limits can increase the practical reproducibility, i.e., the actual and problem-free repeatability of the experiment [28].

In the field of computer-based data-driven science (data science), researchers today often use free and open-source tools and libraries [29]. The reproducibility and repeatability of research results and the description of the specific runtime environment in which the results were generated are not described in the respective publications or only in text form [29] [30]. An important factor in the publication of scientific work is the reproducibility of the research results [30] [31]. It

is therefore necessary that deterministic environments are available to a data scientist. Determinism can be spoken of when all events, especially future events, are clearly defined by preconditions. In other words, development environments are specified accordingly and work as expected, although the methods or algorithms used cannot deliver deterministic results [32].

In data-driven research projects, the application of appropriate data management procedures helps to maintain the data integrity of digital data generated in the research process ("research data"). The preservation of data integrity in the data flow can be maintained by documenting the data origin and applied transformations during the research process. Data is considered reliable if results and errors in the data creation and data analysis process can be reproduced through traceability (good data provenance) [33].

Data provenance is encapsulated by so-called research data management, among other things. Research data management includes all measures to ensure the usability and reusability of research data before, during and after the research project [34]. Systematic representation of these points in the project life cycle is a Data Management Plan (DMP), which describes the data and how the data is processed in the project [35]. A data life cycle illustrates the steps from collection to re-use (creation, preparation, analysis, archiving, access, re-use) [36]. There are more complex models, such as the Curation Lifecycle Model (DCC), which describes various fields of activity in the preservation and maintenance of data [37]. A DMP is required by funders when submitting a proposal (see DFG Form for the continuation of a Collaborative Research Center, 1.4.3, [38]) and serves to ensure effective research data management and long-term usability of the data [34]. Various retention periods apply in order to preserve the reusability of research data. According to good scientific practice, primary data should be stored permanently in research institutions for at least ten years, together with clear and comprehensible documentation of the methods used (e.g., laboratory books) (see Recommendation 7 [39]).

In addition to the requirements arising from research data management, the data principles published in 2016, which define the basis for research data and research data infrastructures to ensure sustainability and reusability, must be taken into account [40]. It was defined by researchers, financiers, publishers and university representatives to increase the reusability of research data (FORCE 11 group). Scientific data should therefore be searchable, accessible, interoperable and reusable. The FAIR data principles can be applied to the entire data life cycle and, as an extension to research data management, provide a collection of best practices for sharing data under ethical and contractual conditions (copyrights, intellectual property rights, etc.).

Highly simplified, data and services should be stored in central data repositories using appropriate metadata (F), taking into account aspects of long-term archiving (A), and should be able to be exchanged and interpreted (semi-)automatically (I) and thus be comparable and reusable (R). If research data cannot be published due to legal requirements, the FAIR principles provide procedures for publishing a description to make the underlying data more understandable. Metadata on machine or human-readable interpretability facilitate comprehensibility and data processing (see Dublin Core Metadata Initiative, Data Documentation Initiative, etc.), open data formats facilitate interchangeability and reusability, metadata on privacy and copyright regulations facilitate accessibility, persistent identifiers assist in finding and easily accessing the information, the indication of licenses (e.g., Creative Commons, etc.) specify the type of usability of the data [41]. Numerous tools support the work with FAIR-Data [42].

In the context of data-driven science, several concepts that place demands on an ecosystem of IT systems must therefore be considered. Following the OSEMN process, the data life cycle of research data management and the FAIR data principles, a schematic summary is shown in Figure 1.

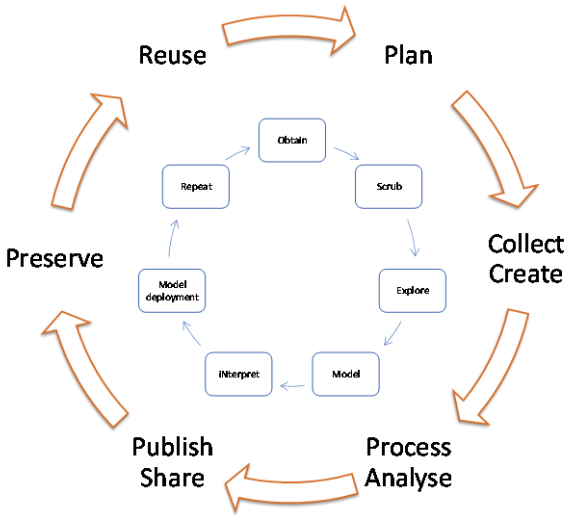


Figure 1. A schematic summary of OSEMN process, the data life cycle of research data management and the FAIR data principles (Source: own illustration).

In addition, depending on the subject area, further guidelines are to be considered. For the secondary data analysis of health data, these are, among others, the "Good Practice in Secondary Data Analysis (GPS)" [43] and the "Good Clinical Practice of the International Council for Harmonisation of Technical Requirements for Pharmaceuticals for Human Use (ICH E6 GCP) [44][45].

IT operators at research institutions should therefore work together with researchers and libraries on IT infrastructures to support the above points. Results of systems research show that open source tools in particular are suitable for the requirements of reproducibility. Although Docker was introduced primarily for business needs and the isolation and deployment of web applications in so-called containers, it provides solutions for virtualization, platform portability, reuse, sharing, archiving and versioning [17] for the scientific community.

The use of tools such as JupyterNotebooks (jupyter.org) enables semantically interoperable publishing of program code, including through the use of the IPYNB format [29]. JupyterNotebooks supports workflows in scientific computing, astrophysics, geology, genetics and computer science [29]. Various applications and programming languages (e.g., Python, R) provide interfaces to JupyterNotebooks [30][46]. Jupyter collects many valuable tools that are needed in the steps of the OSEMN process model.

C. The aims of the project

The aim of this project is to create an easy-to-maintain and cost-effective consolidated IT infrastructure to support data-driven research and implementation in the existing data center infrastructure at the Center for Information Management (ZIMt) at Hannover Medical School (MHH). The requirements for IT systems of the known process models, research data management, FAIR and the operation of applications on enterprise level (such as data security and system recoverability) and numerous other standards, guidelines, directives and recommendations (such as Good Research Practice, Good Practice for Secondary Use of Data, etc.) have to be met.

The ZIMt centralizes operative systems and is a service provider especially for the areas of research and teaching, clinic and administration. IT services (applications) are used in clinical operations to optimize clinical processes and legal documentation and place high demands on system availability and fail-safe IT services. This has an impact on the processes and IT systems of the computer center. In addition, simple interfaces are provided for end users (nursing, doctors, administration) for problem presentation and reporting, which enable centrally controlled fault clearance via an IT service desk. In order to guarantee interference suppression, high demands are placed on the standardization of IT processes and system documentation. The ZIMt operates a class 3 [47] data center at the MHH to guarantee these requirements and is certified according to ISO 9001:2015. In the area of research, however, the IT process landscape is sometimes disruptive, as rapidly changing requirements and moving targets are sometimes necessary to achieve the research goal. The centralization of applications to support the scientific sector is a strategic goal of the MHH.

We have defined three main areas of focus which are to be given special consideration in this proof of concept draft: (1) usability to evaluate the integratability of the IT solution into the system landscape established at the MHH and the working environment familiar to the end user; (2) disaster recovery to evaluate the recoverability of the proposed solution. By integrating a system into the MHH environment, the system has to meet special requirements (front-end branding to maintain corporate identity and security aspects). The user administration and access authorization (3) should be used centrally via an existing directory service ("Active Directory") and security groups defined therein, and must therefore be evaluated. Requirements on the usability of the service can in turn have an influence on the security factors.

These dependencies should therefore be emphasized in this concept. For easy control of used storage resources, the available storage system of the data management provider NetApp [48] was applied. Available interfaces should be used and thus not require any additional effort in the management and monitoring of the system for the IT operator.

This paper is an extended version of our previous work [1] and the further course of this paper is structured as follows. In Section II we describe the methods used to meet the above challenges. In Section III we describe the results achieved in relation to the main topics. Section IV concludes this paper, addressing open questions and the next steps.

II. METHODS

As an interactive shell for various programming languages, JupyterNotebooks is provided as development environments at the MHH. The solution is browser-based and the end user does not need to install any additional software on his device [49]. The open source environment JupyterHub Notebook Server is used to operate JupyterNotebooks in the data center. It enables users to access computer environments and resources without bothering them with installation and maintenance tasks.

Standardized environments (the Docker software/binaries that run on many different operating systems today) built for containers are suitable for using individual application manifests in different locations (e.g., locally as well as in a public cloud infrastructure provided by Google, Amazon Web Services (AWS) or Microsoft Azure) without having to change the code. JupyterHub uses Docker as the basis for the deployment of JupyterNotebooks. The Jupyter Docker Stacks project [50] provides standardized JupyterNotebook environments for various applications using Docker images, including preconfigured environments for use in data science. For special requirements of the development environment, additional images can be offered that can be individually adapted to the user's needs.

In Kubernetes, several containers (e.g., Docker Containers) with common memory and network (common context) can be defined and delivered in a so-called capsule ("Pod") as a coherent structure [51]. For more design flexibility, a hypervisor (VMware) was used to provide the hosts for container orchestration based on Dockers and Kubernetes. Snapshots on VMware enable point-in-time copies of virtual disks can be used to switch to a virtual machine state at an earlier time.

Terraform and Ansible are fully automated. Terraform is an open-source tool that offers the possibility to describe infrastructure configurations programmatically as code (CaaS [52]) (Hashicorp Configuration Language) [53]. As an open-source tool, Ansible provides automation tools for orchestration, general configuration (e.g., software distribution) and administration of IT infrastructures [54]. Terraform creates the virtual machines within the hypervisor, Ansible takes care of the installation of the packages and the configuration of the hosts (configuration management). To

avoid configuration inconsistencies, the DevOps (Software Development and Information Technology Operations) paradigm is considered an "immutable" infrastructure where every change in the ecosystem leads to a completely new deployment of the entire stack [55].

A JupyterHub is a multi-user server for JupyterNotebooks, consisting of several applications that provide different services (hub, notebooks, proxy, authenticator, spawner), which allow secure access to central computing environments and resources.

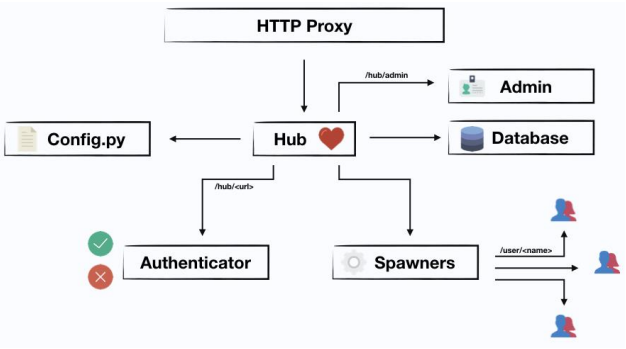


Figure 2. Overview of JupyterHub components [56]

The most important components and their interaction can be briefly explained using the JupytherHub architecture (see Figure 2). The JupyterHub proxy forwards the user to the JupyterHub or to the user's existing JupyterNotebook, depending on the user's sign-on status (SIGNED IN/OUT USER). Existing JupyterNotebooks are managed in the Hub and after successful authentication new JupyterNotebooks as well as a defined user storage (Pods + Volumes) are provided and registered in the proxy. The Hub Container of the JupytherHub as a central component contains three services (Authenticator, User Database, Spawner), which on the one hand can be adapted to the personal requirements and on the other hand are particularly worth protecting [56]. The so-called "Authenticator" authenticates the users against a directory service, in our case a Microsoft Active Directory using the Lightweight Directory Access Protocol (LDAP), and provides the required input mask as a web page. The "User Database" stores login and JupyterNotebook information of the respective users. This data is required for operation and recovery [57]. It is recommended to replace the standard database (SQLite) in a productive environment with a classic relational database management system (RDBMS) such as PostgreSQL or MySQL. The spawner provides the notebooks, is able to communicate with the Kubernetes API and creates ("spawned") Docker Containers. The spawner can be parameterized, so that resources can be limited or a special image can be passed to create the containers via Kubernetes [58][59].

For the proper implementation and configuration of all components, in addition to a manual deployment, helm as used, a package manager for easy installation, publishing, administration, updating and scaling of preconfigured Kubernetes applications. All configurations and communications between the containers can be predefined, as well as the accessibility of the end users from outside. In this project we used a predefined Helm-Chart [60] for the deployment of JupyterHub on Kubernetes. Any changes to the predefined default values can be overwritten by a configuration file during deployment and thus be individually adapted to the requirements in the respective system environment.

Since Docker Containers do not persist data after their life cycle has ended, the storage of configuration and user data must be guaranteed. You must therefore mount persistent volumes to retain the data beyond the life cycle of the container. For this purpose, Kubernetes offers Persistent Volumes, which are usually stored on the nodes of a Kubernetes cluster [61]. In the MHH data center, a high-performance and highly available NetApp storage system is used as the central storage system for consolidating and storing data. To prevent data from being stored on individual distributed devices, the data from the Kubernetes containers should be persisted on this central storage. For this purpose, persistent volumes were provided automatically and centrally via an open source for Kubernetes from NetApp, Trident [62]. Trident offers an ideal interface between persistent volumes and the containers or Pods for this purpose. For each Pod, a separate volume was created on the storage cluster, which by default is only provided for one Pod (e.g., JupyterNotebook). Using so-called storage classes, NetApp storage can map various guidelines for quality of service level or back-up guidelines, among other things, in order to differentiate resource allocation between storage services for students (short availability, balanced performance, daily back-ups) and researchers (long availability, high performance, high back-up frequency).

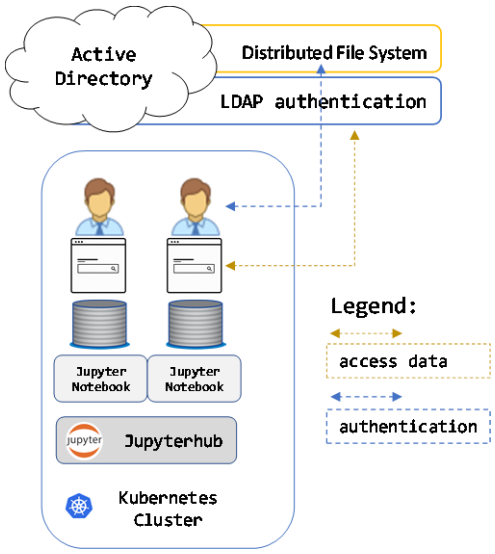


Figure 3. Sketch access to network drive on DFS from a JupyterNotebook context (Source: own illustration).

The workstations at the MHH are equipped with Microsoft Windows by default. The operating mode is strongly influenced by the look and feel of the graphical user interfaces, and project documents are stored on network drives provided by the data center. Microsoft enables directories distributed on different data storage devices in the network to be combined into directory structures via the Distributed File System (DFS). The Server Message Block (SMB) protocol is used to map the file system authorizations with authorization objects of the Active Directory in the network (via DFS) (see Figure 3) [64]. In JupyterHub the user's network drives are not available in raw state, but he must be able to store raw data, the generated program codes and result files from and to the network drive where the central access (also of the team members) takes place.

The used JupyterHub container is based on a Linux operating system (Ubuntu). A user can mount SMB shares in the available directory structure under Linux using the LinuxCIFS utils package [64]. For the use of LinuxCIFS utils increased system rights are required. This requires a corresponding implementation within the container, but Docker also implements standard security rules and thus, for example, prohibits the execution of the command for mounting network drives in the default settings, which in Kubernetes is done via the so-called "privileged mode" [65] [66]. For security reasons, applications should only be given the most necessary privileges (see chapter 6 [67]). To enable the privileged mode, the corresponding configuration parameters were transferred via the Helm-values-file (config.yaml). The LinuxCIFS utils are not included in the used Dockerfile [68] of JupyterNotebook of this JupyterHub Helm chart. We have manipulated the Dockerfile accordingly, and integrated the LinuxCifs utils. In addition, the user under which the JupyterNotebook is executed ("jovyan") needs increased rights of the superuser (so-called "root") to execute the command. The assignment of the execution rights is done via the "sudoers" file. The user jovyan has only been granted rights to execute the following modules (mount/umount). Then the Docker image was built with this customized Dockerfile and deployed on each Kubernetes worker node. The JupyterHub Helm chart with a customized helm-values-file (config.yaml) was installed to use the new Docker image [69]. To give the end user the familiar look and feel of the corporate environment, it is necessary to customize the application according to the corporate design of MHH. On the log-in page, the user receives the company logo so that there is direct recognition value to an in-house application. For this purpose, an adapted version of the HTML files for the log-in page of JupyterHub was provided on the Kubernetes workstations. The files are located in the container of JupyterHub in the Unix path "/usr/local/share/jupyterhub/templates" and were adapted to a path on the Kubernetes-Worker using helm-values. Kubernetes controls this process automatically. The log-in pages can thus be adjusted at runtime.

Even though we are in a "proof of concept" phase of the project, we wanted to integrate authentication methods from the beginning to control access to the platform while achieving security compliance. At the MHH, a local security area for managing objects (e.g., user names, computers, printers, etc.) is implemented as a domain via Microsoft Windows Active Directory. To centralize the administration of user IDs using Role-Based Access Control (RBAC), authentication to the Active Directory was accessed using JupyterHub's Lightweight Directory Access Protocol (LDAP) implementation. A further step towards simplifying the login to a service, centralizing authentication and ensuring the company's password policies was the integration of user authentication via LDAP. For this purpose, the section for the Authenticator in the Helm-values was adapted. A special security group was defined in the Active Directory to restrict the user circle and to allow a control of user releases for this service.

The user can decide via different spawners which Docker image and thus which standard runtime environment and amount of resources should be provided. This was also realized via the Helm-values. With preconfigured spawners, the user does not have to configure the runtime environment every time he/she starts the environment, e.g., to get an integration of special packages. A selection of preconfigured Dockerfiles is available on the Docker-Hub [70].

Due to the central storage of the runtime environment and the strict alignment of the project to CaaS, a quick recovery of the service is easily possible. The following measures were taken to achieve this:

1. creating the Kubernet cluster and separate needed virtual machines via Vagrantfiles
2. configuration of machines using Ansible Playbooks
3. restore the Kubernetes database including Trident configuration [19]
4. restore JupyterHub using Helm-chart and persistent volumes

JupyterHub is a central element in the OSEMN process model. It can be used for data preparation, feature extraction and model programming in the steps Scrub, Explore and Model. We recommend the use of the openData Platform CKAN [71] for the acquisition and administration of data sets. This modern UI enables easy navigation and search for available data sets from other or own projects using suitable metadata in accordance with the FAIR principles (see, e.g., [72]). CKAN is thus a possible technological component in the Obtain process step. CKAN also offers the possibility to store the files in an object storage via an Amazon S3 interface [73]. During the entire data science process, it may be necessary to load, process or store data or artifacts from different sources. Especially for different "unstructured" data, the use of object storage as data storage is recommended, since the administration of the objects by means of metadata provides a higher flexibility and context description of the data sets than when storing them on a common file system. To manage the contents in object storage, we recommend the use of Minio. Minio implements interfaces to Java, Go, Node.js, Python and .NET [74] and is therefore well suited for the most common programming languages in the Data Science environment [75]. For structured data and using the Python libraries pandas [76] and SQLAlchemy [77] a database management system

(DBMS) is recommended. An OpenSource solution of a DBMS is Postgres [78]. In the steps Explore and Interpret it is helpful to create visualizations of the data. A suitable open source tool for this can be Superset [79]. Superset is a dashboarding tool that is easy to use with little technical know-how and offers a variety of ready-made and interactive visualizations. As interfaces with databases Superset implements the well-known JDBC or ODBC drivers [80].

The technological components used enable the monitoring and evaluation of the health status of services using various metrics. The number of running Pod's under Kubernetes gives information about the indirect number of users, the users logged on the system. The utilization of individual Pods is possible via the Kubernetes service "heapster" [81]. The latency is a measure to evaluate the reaction time between application and client (end user). A monitoring of the response time can be realized by different methods. In our case, the central IT monitoring software "Checkmk" [82] is used, which allows to monitor different metrics of a device. This way, besides latency, other essential metrics such as memory usage and resource utilization (CPU/RAM) can be monitored. With the help of this monitoring it is possible to react proactively to upcoming problems. In addition, by storing the monitoring data, long-term analyses and trends can be identified, which can be used to plan the expansion of the environment.

Kubernetes exposes interfaces to Kubernetes management and cluster control in its own network segment. To prevent users from the corporate network from accessing these management interfaces, the Kubernetes Cluster was configured with a separate network having its own IP range. Via a so-called reverse proxy [83] we enable services from the network segment of the Kubernetes Cluster to be exposed in the corporate network (separate segment). The reverse proxy accepts requests from the company network and forwards them (depending on the given address), e.g., into the network of the Kubernetes Cluster. In this way this proxy provides, e.g., for JupyterHub centrally the URL to access the platform and extends the environment by an encrypted data transfer between end user and the JupytherHub Proxy, using the Transport Layer Security (TLS) [84]. When providing Pods on Kubernetes, services are bound to the IP address of the possible Kubernetes node via ports. By default, these IP addresses and ports are assigned dynamically at the time of provision [85]. We have assigned each service via a specific port according to the LoadBalancer principle [86][87]. This way the reverse proxy can reach the service at any time in the Kubernetes Cluster. Kubernetes takes care of the failover thanks to the integrated High Availability functions using LoadBalancer.

III. RESULTS

Based on the methods described in II, an ecosystem consisting of the proposed software components for mapping services was implemented. Using JupyterHub as an example, the levels at which such a service must be integrated into an enterprise in order to meet the requirements placed on an IT service provider were illustrated.

The operator of the infrastructure (ZIMt) achieves a reduction of workload through the chosen reference architecture (see Figure 4) by automating the provision of resources for the users (researchers) and minimizing the effort to provision resources for research purposes.

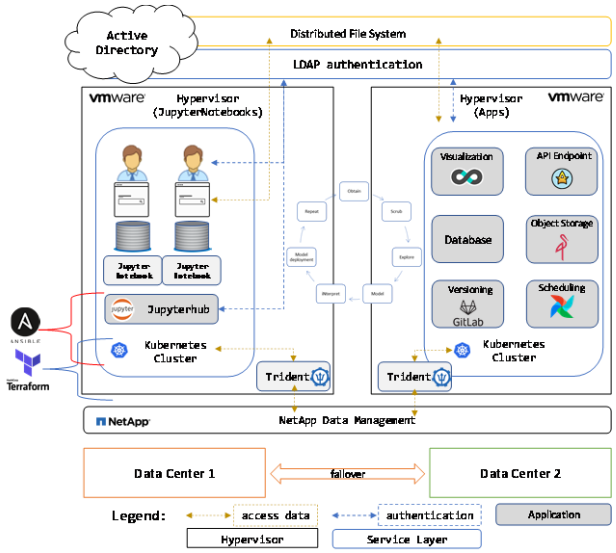


Figure 4. Prototypical architecture for deploying JupyterNotebooks on enterprise technology. To the centrally indicated OSEMN process, the Kubernetes node for the JupyterHub infrastructure is shown on the left. On the right, another Kubernetes node is shown with additional exemplary tools to support the OSEMN process. The components are in failover to a second data center (see bottom) (Source: Own illustration).

The prototypically implemented infrastructure enables the end user (students, scientists) to easily use JupyterNotebooks. With the Docker-based approach, the description of the runtime environment required for the research approach can be fixed using the Docker-specific tagging facility and stored in a manifest for publication in a comprehensible and interoperable manner [30].

JupyterHub allows users to interact with a computing environment via a web page. Since most devices have access to a web browser, JupyterHub makes it easy to deploy and standardize the computing environment to a group of people (for example, a class of students or an analysis team). Additional tools (CKAN, Postgres, Minio, Superset) are also accessible via a web interface and provide additional programming interfaces that can be addressed in JupyterNotebook. Furthermore, it could be shown that Docker images of the JupyterNotebooks, especially adapted to the specific requirements of the company, could be created and made available for selection via the JupyterHub Spawner (Basic and Data Science). Additional libraries or tools, which must not be included in the standardized environment, can be flexibly installed in a separate area within the current runtime environment. JupyterNotebooks thus offer the possibility to use several tools without changing the environment. Requirements for different tools, such as those

needed in processes like OSEMN, can be mapped with the software products mentioned above (see Table I).

TABLE I. MAPPING OSEMN PROCESS PHASES AND PROPOSED TECHNOLOGY

OSEMN Phase	Domain	Proposed Technology
Obtain	Data Search	CKAN
	Data I/O	Postgres
		Minio
Scrub	Data Transformation	Jupyter
	Data I/O	Postgres
		Minio
Explore	Pattern Finding	Superset
	Feature Extraction	Jupyter
	Data I/O	Postgres
		Minio
Model	Modeling	Jupyter
	Data I/O	Postgres
		Minio
iNterpret	Review	Superset

If the researcher chooses a working environment based on JupyterNotebooks, necessary work steps and results can be stored together with the notebook [29][46]. The basic requirements for the implementation of requirements from research data management and the FAIR principles can be fulfilled. A notebook acts as a laboratory book and describes the steps of data processing. The GitLab also provides a history function, so that the researcher can ensure data provenance in the research project. The isolation of a researcher's specific work area can therefore be achieved by using container technology. Kubernetes offers the possibility to consolidate central computing resources in a data center and to use them efficiently due to the integrated load distribution and error bypass.

It could be shown that already available IT infrastructure could be sensibly integrated (including storage systems, hypervisor infrastructure, management, monitoring). The connection of the components to the company's own IT monitoring system consolidates the various metrics in one place and facilitates the management of the prototypically implemented infrastructure.

Since JupyterHub is provided via Dockers, branding requirements to maintain corporate identity can be easily met. It was shown that the login page of JupyterHub could be adapted to the corporate design of the company and, among other things, be branded with the company logo. By using the existing Active Directory, user access can be managed centrally. Authentication via LDAP simplifies the login to the system, since no separate access data has to be maintained and incorrect logins can be registered and used to block the user account in case of attempted misuse.

Each time a user logs on to the start page, the system checks whether the user already owns a JupyterNotebook created in the past. In this case he can be redirected to his previously created environment. Otherwise, a new notebook (in the form of a new container) is created and new storage space is provided (because this user did not exist before). By providing the required storage space at runtime, resources can be provided centrally and efficiently. The persistence of research data outside the Docker Container runtime on the existing enterprise storage system could be solved efficiently by using Trident (see Figure 4). This means that the processing steps required during the process (including temporary ones) can be performed on a high-performance storage system that can guarantee the persistence of research data in any case. Nevertheless, the user is able to make the final script files and data of the project available to other members of the department via the mountable department drive. The department drive (DFS) (see Figure 4) can thus be reused in its original function and offers researchers without technical affinity the possibility to access the project data via their usual working methods.

IV. DISCUSSION

In this paper we show a prototypical implementation for the efficient use of available data center resources as a self-service platform on enterprise technology to support data-driven research.

Although the OSEMN process is a suitable, easy-to-understand reference, there are some extensions that are proposed below. A major drawback of the OSEMN process is that it is understood as a linear, aperiodic process. Compared to other established process models for data analysis/data science, such as Knowledge Discovery in Databases (KDD) [88] or Cross Industry Standard Procedure for Data Mining (CRISP-DM) [89], the knowledge gained is not played back and (at least formally) no iterations take place. However, this re-iteration is a decisive step, since many projects are more successful due to their exploratory character [90], if they are carried out in short iterations. In the course of these cycles many of the tasks are repeated, such as data cleansing or training of models. In order to use the available resources as effectively as possible, it is recommended to aim for the highest possible degree of automation (extension by the Repeat component). Possible tools for this would be Apache Airflow [91] or the Python library Kedro [92]. Another crucial step, which is included in the CRISP-DM model in contrast to OSEMN, is the deployment of the developed (ready-to-use) model (see "Phase 6 - Deployment" in CRISP-DM). This allows the trained models to be decoupled from the underlying technology (e.g., Python, R, Julia) and made available to a wider audience via standardized web interfaces (REST via HTTP). Examples are Python libraries like Flask [93] or HUG [94]. When providing models, the aforementioned automation aspect has quality assurance features. The "decay" of a model can be detected and corrected proactively by regular and automated testing of the endpoints (see Model Decay & Concept Drift [95]). We therefore propose to extend the OSEMN process model after the iNterpret step by the steps Model Serve/Deploy ("Mo") and Repeat ("Re") transversal to OSMEN (see Table II).

Kubernetes was used as an open source solution to orchestrate, automate and fulfil these high availability requirements for the container-based infrastructure [96]-[98]. It is a widely used and proven technology for providing services like Jupyter. Since Kubernetes is designed to host a huge number of containerized applications with minimal overhead, it is perfectly suited for many JupyterNotebooks and other potential applications within the science ecosystem [99][100].

TABLE II. MAPPING OSEMN PROCESS PHASES AND PROPOSED TECHNOLOGY

MoRE OSEMN Phase	Domain	Component	Proposed Technology
Model Serve & Deploy	Deployment	API End Point	API Star
	Tracking	Usage Monitoring	CheckMK
		Model Monitoring	Superset
Repeat	Automation	Scheduling Engine	Airflow

For the prototypical implementation of this infrastructure a Kubernetes master with two Kubernetes nodes ("Worker") was used. For productive operation, at least two Kubernetes Masters should be used in order to meet the requirements for fail-safe operation. In the event of a disaster recovery scenario and the loss of the entire Kubernetes cluster, the storage volumes provided via Trident must be reconnected manually. An automatism for recovery procedures would still have to be created. In an emergency, the administrator can migrate the contents of the corresponding volume using an NFS interface.

Docker Containers are more convenient to implement, easier to manage, minimize the overhead of resource usage, and are therefore more efficient than traditional virtual infrastructures [99]. The provision of environments (e.g., by containers) in the academic sector can be very large, thus increasing the burden on the operators and maintainers of the environment. Automating the deployment and orchestration of the environment is strongly recommended. Running applications in containers does not automatically solve the challenge of protecting these applications from outages (such as hardware failures or resource bottlenecks on the one server we are working on). Even if containers are encapsulated on an operating system, there may be problems with the underlying host on which the container is running - therefore an additional software layer is required to take care of resource planning and availability of any services.

The use of the predefined configurations of the JupyterNotebooks is initially limited by the Docker Images. If the service is used for a longer period of time, it will become apparent whether the provision of additional images makes sense.

The authorization of the user in the JupyterNotebook Docker Container to mount SMB network drives by sharing via the sudoers file and the activation of the increased privileges in Kubernetes inevitably leads to serious security vulnerability. The user would be able to provide his own file system and replace the sudoers file with a specially manipulated file. The consequence would be system administrator privileges. The solution used must therefore be adapted taking security aspects into account. For example, Kubernetes could be enabled via the vSphere API [https://github.com/kubernetes-sigs/cluster-api-provider-vsphere] to provide each JupyterNotebook-Pod in an isolated virtual machine, which would result in a stricter isolation of the Docker Containers from each other.

In order to make it as easy as possible for end users to transfer the artefacts created in the JupyterNotebook to the project repositories on the network drives, the end user should be able to integrate the network drive into his JupyterNotebook environment. We expect a higher user acceptance despite the use of new technology / application.

A Pod (container), in which a JupyterNotebook is running, expires after a certain time without activity (max idle time) [101], so that the resources can be released again and used for other users / Pods (downscaling). If a Pod expires, changes made at runtime are also discarded, since they are not part of the Docker image that is called by JupyterHub every time a Pod is initialized. The user must therefore remount his network drives after such a reset.

Existing and established authentication methods such as OAuth [102] or OpenID [103] offer additional flexibility, as users outside of the Active Directory could be integrated. The necessary components were not available during the project. However, these security concepts will be considered in later phases of the project after the validation of the first thesis (if this software stack is suitable for the use case at all).

Despite monitoring and a high degree of automation of the individual system components, errors can occur during operation. These can be of different nature, but can be roughly divided into logical and physical errors. Logical errors in data lead to inconsistencies and physical (hardware) errors are associated with data loss. As a countermeasure, the system components are protected by creating backups. For this purpose, system copies - so-called "snapshots" - are created at regular intervals by the hypervisor or storage technology used, which can be accessed as required. If the JupyterHub internal database is lost, the connection to the Pod and thus to the individual runtime environment is lost and must be restored.

The presented proof of concept could demonstrate the feasibility of IT operations by combining common data science tools with the enterprise architecture. In the next development stages, tools such as Airflow [91] or Pachyderm [24] (as a platform solution) for pipelining and automation can be used in addition to JupyterNotebooks in connection with machine learning. These tools could support process models like OSEMN as well as aspects of reproducibility and reusability. Integration of tools specifically for big data use cases is not recommended, as these may require special ecosystems (see [104]). More often than not, the end user can already interact with available big data platforms at any time using the programming languages available in JupyterNotebooks.

REFERENCES

[1] S. Guhr et al., "Data Science as a Service - Prototyping for an Enterprise Self-Service Platform for Reproducible Research", IARIA - The Fifth International Conference on Fundamentals and Advances in Software Systems Integration (FASSI 2019), 2020

[2] Landesarbeitskreis Niedersachsen für Informationstechnik/Hochschulrechenzentren, "Landes-IT-Konzept Hochschulen Niedersachsen", 2018, https://www.lanit-hrz.de/fileadmin/user_upload/Landes-IT-Konzept_Hochschulen_Niedersachsen_2019-2024.pdf, last accessed 2020/02/27

[3] V. Kale, "Big Data Computing: A Guide for Business and Technology Managers", Chapman and Hall/CRC, 2016, ISBN: 978-1498715331

[4] W. C. Preston, "Does Object Storage kill RAID?", Storage Switzerland, 2016, https://storageswiss.com/2016/02/09/does-object-storage-kill-raid/, last accessed 2020/02/27

[5] J. Hernantes, G. Gallardo, and N. Serrano, "IT Infrastructure-Monitoring Tools," IEEE Software, vol. 32, no. 4, pp. 88-93, 2015. DOI: 10.1109/MS.2015.96

[6] The International Foundation for Information Technology, 2009, https://www.if4it.com/SYNTHESIZED/GLOSSARY/S/System_Integration_Management_Service_Level_Agreement_SLA.html/, last accessed 2020/02/27

[7] C.-P. Praeg and D. Spath, "Quality Management for IT Services: Perspectives on Business and Process Performance (Advances in Logistics, Operations, and Management Science)", Business Science Reference, 2011, ISBN: 978-1616928896)

[8] M. A. Vonderembse, T. S. Raghunathan, and S. Subba Rao, "A post-industrial paradigm: To integrate and automate manufacturing", International Journal of Production Research, 35:9, 2579-2600, 1997, DOI: 10.1080/002075497194679

[9] C. Li, "Preprocessing Methods and Pipelines of Data Mining: An Overview", CoRR, 2019, arXiv:1906.08510

[10] H. Mason and C. Wiggins, "A Taxonomy of Data Science", dataists.com, 2010, http://www.dataists.com/2010/09/a-taxonomy-of-data-science/, last accessed 2019/07/22

[11] Python Programming Language, 2019, https://www.python.org, last accessed 2020/02/23

[12] The R Project for Statistical Computing, https://www.r-project.org, last accessed 2020/02/23

[13] SAS® Enterprise Miner™, 2019, https://www.sas.com/en_us/software/enterprise-miner.html, last accessed 2020/02/23

[14] Open Refine, http://openrefine.org/, last accessed 2020/02/23

[15] Pandas Python Data Analysis Library, https://pandas.pydata.org/, last accessed 2020/02/23

[16] Scipy, 2019, https://www.scipy.org/, last accessed 2020/02/23

[17] Rapid Miner, 2019, https://rapidminer.com/, last accessed 2020/02/23

[18] KNIME End to End Data Science, 2019, https://www.knime.com/, last accessed 2020/02/23

[19] scikit-learn: machine learning in Python, https://scikit-learn.org/stable/, last accessed 2020/02/23

[20] A Short Introduction to the caret Package, https://cran.r-project.org/web/packages/caret/vignettes/caret.html, last accessed 2020/02/23

[21] matplotlib, 2019, https://matplotlib.org/, last accessed 2020/02/23

[22] Tableau, 2019, https://www.tableau.com, last accessed 2020/02/23

[23] Microsoft Power BI, 2019, https://powerbi.microsoft.com, last accessed 2020/02/23

[24] Pachyderm - Reproducible Data Science that Scales!, 2019, https://www.pachyderm.io/, last accessed 2020/02/23

[25] A. Woodie, "Inside Pachyderm, a Containerized Alternative to Hadoop", datamai, 2018, https://www.datanami.com/2018/11/20/inside-pachyderm-a-containerized-alternative-to-hadoop/, last accessed 2020/02/23

[26] Max Planck Society, "Rules of Good Scientific Practice", 2000, https://www.evolbio.mpg.de/3306231/rules-of-good-scientific-practice.pdf, last accessed 2020/02/28

[27] J. B. Buckheit and D. L. Donoho, "WaveLab and Reproducible Research", Wavelets and Statistics, pp. 55-81, 1995, DOI: 10.1007/978-1-4612-2544-7_5

[28] M. Bussonnierk et al., "Binder 2.0 - Reproducible, interactive, sharable environments for science at scale", Conference: Python in Science Conference, 2018, DOI: 10.25080/Majora-4af1f417-011

[29] T. Kluyver et al., Jupyter Development Team, "JupyterNotebooks – a publishing format for reproducible computational workflows", IOS Press. pp. 87-90, 2016, DOI: 10.3233/978-1-61499-649-1-87

[30] C. Boettiger, "An introduction to Docker for reproducible research, with examples from the R environment", ACM SIGOPS Oper. Syst. Rev.. 49. 10.1145/2723872.2723882. https://arxiv.org/pdf/1410.0846.pdf

[31] Nature Editors 2012. "Must try harder". Nature. 483,7391 Mar. 2012, 509–509.

[32] D. L. Donoho, "An invitation to reproducible computational research", Biostatistics, Volume 11, Issue 3, July 2010, pp. 385–388, DOI: 10.1093/biostatistics/kxq028

[33] E. Boose et al., "Ensuring reliable datasets for environmental models and forecasts", Ecological Informatics 2(3):237-247, 2007, DOI: 10.1016/j.ecoinf.2007.07.006

[34] K.F. Holmstrand, S.P.A. den Boer, E. Vlachos, P.M. Martínez-Lavanchy, and K.K. Hansen, "Research Data Management (eLearning course)", Eds., 2019. doi: 10.11581/dtu:00000047

[35] Wikipedia contributors, "Data management plan", Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Data_management_plan&oldid=918458183, last accessed 2020/02/29

[36] UK Data Service, "Research data lifecycle", https://www.ukdataservice.ac.uk/manage-data/lifecycle.aspx, last accessed 2020/02/28

[37] Digital Curation Centre University of Edinburgh, "DCC Curation Lifecycle Model", http://www.dcc.ac.uk/resources/curation-lifecycle-model, last accessed 2020/02/28

[38] Deutsche Forschungsgemeinschaft (DFG), "Antragsmuster für die Fortsetzung eines Sonderforschungsbereichs", 2019, https://www.dfg.de/formulare/60_200/60_200_de.pdf, last accessed 2020/02/28

[39] German Research Foundation (DFG), "Safeguarding Good Scientific Practice", 2013, DOI: 10.1002/9783527679188.oth1

[40] M. D. Wilkinson et al., "The FAIR Guiding Principles for scientific data management and stewardship", Scientific Data 3, doi : 10.1038/sdata.2016.18, 2016.

[41] P. M. Martínez-Lavanchy, F. J. Hüser, M. C. H. Buss, J. J. Andersen, and J. W. Begtrup, "FAIR Principles, DOI: 10.11581/dtu:00000049

[42] Danish e-infrastructure Cooperation (DeiC), "FAIR for Beginners", https://vidensportal.deic.dk/en/FAIR, last accessed 2020/02/28

[43] E. Swart et al., "Good Practice of Secondary Data Analysis (GPS), guidelines and recommendations), Gesundheitswesen

2015; 77(02): 120-126, Third Revision 2012/2014, DOI: 10.1055/s-0034-1396815

[44] International Council for Harmonisation of Technical Requirements for Pharmaceuticals for Human Use (ICH), "Guideline for good clinical practice", 2016, https://database.ich.org/sites/default/files/E6_R2_Addendum.pdf, last accessed 2020/02/28

[45] G. Tancev, "An Introduction to Clinical Data Science", 2019, https://towardsdatascience.com/clinical-data-science-an-introduction-9c778bd83ea2, last accessed 2020/02/28

[46] E. Cirillo, "TranSMART data exploration and analysis using Python Client and JupyterNotebook", 2018, http://blog.thehyve.nl/blog/transmart-data-exploration-and-analysis-using-python-client-and-jupyter-notebook,last accessed 2019/07/22

[47] OVH SAS. 2018 "Understanding Tier 3 and Tier 4". https://www.ovh.com/world/dedicated-servers/understanding-t3-t4.xml, last accessed 2018/09/15.

[48] K. Kerr, "Gartner Named NetApp a Leader in Magic Quadrant for 2019 Primary Storage", 2019, https://blog.netapp.com/netapp-gartner-magic-quadrant-2019-primary-storage/, last accessed 2020/02/23

[49] J. VanderPlas, "Python Data Science Handbook: Essential Tools for working with Data", O'Reilly UK Ltd., 2016, https://jakevdp.github.io/PythonDataScienceHandbook/, last accessed 2020/02/28

[50] Jupyter Docker Stacks, 2018, https://jupyter-Docker-stacks.readthedocs.io/en/latest/, last accessed 2020/02/23

[51] Kubernetes Authors, "Pods", 2020, https://kubernetes.io/docs/concepts/workloads/Pods/Pod/, last accessed 2020/02/28

[52] C. de Botton, "Writing Your Code as a Service, Part I", 2015, https://medium.com/@brooklynfoundry/writing-your-code-as-a-service-part-i-2b960c19ca6e, last accessed 2020/02/28

[53] HashiCorp, "How Terraform Works", https://www.terraform.io/docs/extend/how-terraform-works.html, last accessed 2020/02/28

[54] Red Hat, "OVERVIEW How Ansible Works", https://www.ansible.com/overview/how-ansible-works, last accessed 2020/02/28

[55] P. Debois and J. Humble, "The DevOps Handbook: how to create World-Class agility, Reliability, and Security in Technology Organizations", IT Revolution Press, 2016, ISBN: 978-1942788003

[56] Project Jupyter team, "JupyterHub", 2016, https://jupyterhub.readthedocs.io/en/stable/, last accessed 2020/02/28

[57] M. van Niekerk, "Recovering from a Jupyter Disaster", 2019, https://medium.com/flatiron-engineering/recovering-from-a-jupyter-disaster-27401677aeeb, last accessed 2020/02/28

[58] jupyterhub_config.py, https://github.com/jupyterhub/jupyterhub/blob/master/examples/spawn-form/jupyterhub_config.py, last accessed 2020/02/28

[59] Project Jupyter team, Spawners, 2016, https://jupyterhub.readthedocs.io/en/stable/reference/spawners.html, last accessed 2020/02/28

[60] The JupyterHub Helm chart, https://github.com/jupyterhub/helm-chart, last accessed 2020/02/28

[61] The Kubernetes Authors, "Persistent Volumes", 2020, https://kubernetes.io/docs/concepts/storage/persistent-volumes/, last accessed 2020/02/28

[62] NetApp Trident, 2019, https://netapp-trident.readthedocs.io/en/latest/introduction.html, last accessed 2020/02/23

[63] D. Batchelor and M. Satran, "Microsoft SMB Protocol and CIFS Protocol Overview", https://docs.microsoft.com/en-us/windows/win32/fileio/microsoft-smb-protocol-and-cifs-protocol-overview, last accessed 2020/02/23

[64] Samba.org, "LinuxCIFS utils", 2019, https://wiki.samba.org/index.php/LinuxCIFS_utils, last accessed 2020/02/23

[65] Docker Inc., "Runtime privilege and Linux capabilities", https://docs.Docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities, last accessed 2020/02/23

[66] The Kubernetes Authors, "Pod Security Policies", https://kubernetes.io/docs/concepts/policy/Pod-security-policy/, last accessed 2020/02/28

[67] A. Martin, "11 Ways (Not) to Get Hacked", 2018, https://kubernetes.io/blog/2018/07/18/11-ways-not-to-get-hacked/, last accessed 2020/02/28

[68] Docker Hub, "Minimal Jupyter Notebook Stack", https://hub.Docker.com/r/jupyter/minimal-notebook/, last accessed 2020/02/28

[69] Project Jupyter Contributors, "Setting up JupyterHub", 2020, https://z2jh.jupyter.org/en/latest/setup-jupyterhub/setup-jupyterhub.html, last accessed 2020/02/28

[70] Docker Hub, "jupyter repositories", https://hub.Docker.com/u/jupyter, last accessed 2020/02/28

[71] CKAN Association, "ckan", https://ckan.org/, last accessed 2020/02/28

[72] UK Government, "Find open data", https://data.gov.uk/, last accessed 2020/02/28

[73] Amazon Web Services, "Introduction to Amazon S3, https://docs.aws.amazon.com/en_en/AmazonS3/latest/dev/Introduction.html, last accessed 2020/02/28

[74] Minio, Inc, "minio", https://min.io, last accessed 2020/02/28

[75] IEEE, "Interactive: The Top Programming Languages", 2020, https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2019, last accessed 2020/02/28

[76] pandas, https://pandas.pydata.org/, last accessed 2020/02/28

[77] SQLAlchemy authors and contributors, "The Python SQL Toolkit and Object Relational Mapper", https://www.sqlalchemy.org/, last accessed 2020/02/28

[78] The PostgreSQL Global Development Group, PostgreSQL, https://www.postgresql.org/, last accessed 2020/02/28

[79] The Apache Software Foundation, "Apache Superset (incubating)", https://superset.incubator.apache.org/, last accessed 2020/02/28

[80] J. Görner, "Beyond Jupyter Notebooks", https://github.com/jgoerner/beyond-jupyter, last accessed 2020/02/28

[81] The Heapster contributors, "Heapster", https://github.com/kubernetes-retired/heapster, last accessed 2020/02/28

[82] tribe29 GmbH, https://checkmk.com/, last accessed 2020/02/28

[83] Wikipedia contributors, "Reverse proxy", Wikipedia, The Free Encyclopedia., 2020, https://en.wikipedia.org/w/index.php?title=Reverse_proxy&oldid=937552513, last accessed 2020/02/29

[84] Wikipedia contributors, "Transport Layer Security", Wikipedia, The Free Encyclopedia., 2020, https://en.wikipedia.org/w/index.php?title=Transport_Layer_Security&oldid=943166788, last accessed 2020/02/29

[85] The Kubernetes Authors, "Using a Service to Expose Your App", https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/, last accessed 2020/02/28

[86] The Kubernetes Authors, "Service", https://kubernetes.io/docs/concepts/services-networking/service/#loadbalancer, last accessed 2020/02/28

[87] S. Dinesh, “Kubernetes NodePort vs LoadBalancer vs Ingress? When should I use what?”, https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0, last accessed 2020/02/28

[88] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, “Knowledge Discovery in Databases: An Overview”, AI Magazine, 13(3), 57, 1992, DOI: 10.1609/aimag.v13i3.1011

[89] P. Chapman et al. "CRISP-DM 1.0: Step-by-step data mining guide”, Computer Science, 2000

[90] R. Jurney, “A manifesto for Agile data science”, 2017, https://www.oreilly.com/radar/a-manifesto-for-agile-data-science/, last accessed 2020/02/28

[91] The Apache Software Foundation, “Apache Airflow”, https://airflow.apache.org/, last accessed 2020/02/28

[92] The Kedro contributors, “Kedro”, https://github.com/quantumblacklabs/kedro, last accessed 2020/02/28

[93] A. Ronacher and contributors, Flask, https://palletsprojects.com/p/flask/, last accessed 2020/02/28

[94] https://www.hug.rest/, last accessed 2020/02/28

[95] A. Chilakapati, “Concept Drift and Model Decay in Machine Learning”, 2019, https://towardsdatascience.com/concept-drift-and-model-decay-in-machine-learning-a98a809ea8d4, last accessed 2020/02/28

[96] L. Hecht, “What the data says about Kubernetes deployment patterns”, 2018, https://thenewstack.io/data-says-kubernetes-deployment-patterns/, last accessed 2019/07/22

[97] Project Jupyter Contributors, “Zero to JupyterHub with Kubernetes”, 2019, https://zero-to-jupyterhub.readthedocs.io/en/latest/, last accessed 2019/07/22

[98] S. Conway, “Survey shows Kubernetes leading as orchestration Platform”, 2018, https://www.cncf.io/blog/2017/06/28/survey-shows-kubernetes-leading-orchestration-platform/, last accessed 2019/07/22

[99] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, “A comparative study of Containers and Virtual Machines in Big Data environment”, arXiv:1807.01842v1

[100] S. Talari, “Why Kubernetes is a great choice for Data Scientists”, https://towardsdatascience.com/why-kubernetes-is-a-great-choice-for-data-scientists-e130603b9b2d, last accessed 2019/07/28

[101] Project Jupyter Contributors, “Customizing User Management”, https://zero-to-jupyterhub.readthedocs.io/en/latest/customizing/user-management.html, last accessed 2019/07/28

[102] OAuth community site, https://oauth.net/, last accessed 2020/02/23

[103] OpenID Foundation, 2019, https://openid.net, last accessed 2020/02/23

[104] S. Gerbel, “Implementation of a sustainable IT ecosystem for the use of clinical data to support patient-oriented research”, GI (Gesellschaft für Informatik): Medizininformatik (Medical Informatics), 2020, https://www.researchgate.net/publication/340925601_Implementation_of_a_sustainable_IT_ecosystem_for_the_use_of_clinical_data_to_support_patient-oriented_research, last accessed 2020/05/21