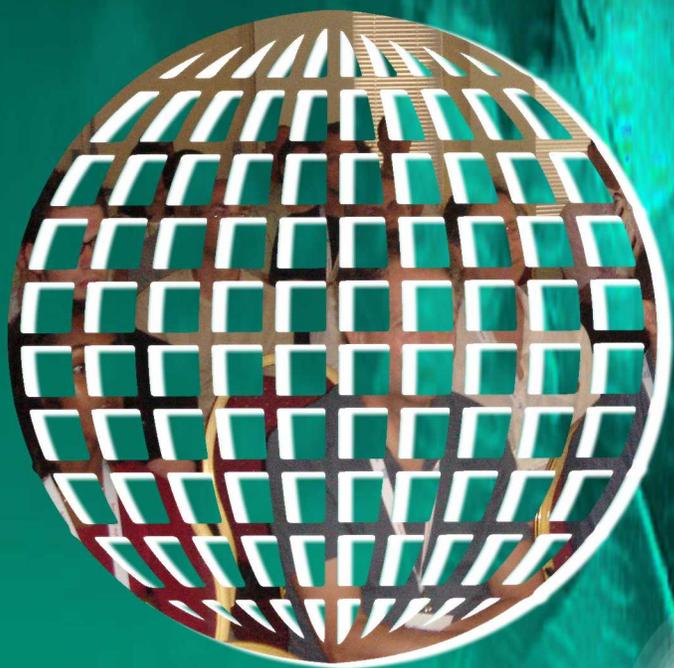


International Journal on

Advances in Software



2012 vol. 5 nr. 1&2

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

ISSN: 1942-2628

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

contact: petre@aria.org

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

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

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

Reference should mention:

International Journal on Advances in Software, issn 1942-2628
vol. 5, no. 1 & 2, year 2012, <http://www.ariajournals.org/software/>

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

Reference to an article in the journal is as follows:

<Author list>, "<Article title>"
International Journal on Advances in Software, issn 1942-2628
vol. 5, no. 1 & 2, year 2012,<start page>:<end page> , <http://www.ariajournals.org/software/>

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

Sponsored by IARIA

www.aria.org

Copyright © 2012 IARIA

Editor-in-Chief

Jon G. Hall, The Open University - Milton Keynes, UK

Editorial Advisory Board

Hermann Kaindl, TU-Wien, Austria

Herwig Mannaert, University of Antwerp, Belgium

Editorial Board

Witold Abramowicz, The Poznan University of Economics, Poland

Abdelkader Adla, University of Oran, Algeria

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

Marc Aiguier, École Centrale Paris, France

Rajendra Akerkar, Western Norway Research Institute, Norway

Zaher Al Aghbari, University of Sharjah, UAE

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

Ahmed Al-Moayed, Hochschule Furtwangen University, Germany

Giner Alor Hernández, Instituto Tecnológico de Orizaba, México

Zakarya Alzamil, King Saud University, Saudi Arabia

Frederic Amblard, IRIT - Université Toulouse 1, France

Vincenzo Ambriola, Università di Pisa, Italy

Renato Amorim, University of London, UK

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

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

Philip Azariadis, University of the Aegean, Greece

Thierry Badard, Université Laval, Canada

Muneera Bano, International Islamic University - Islamabad, Pakistan

Fabian Barbato, Technology University ORT, Montevideo, Uruguay

Barbara Rita Barricelli, Università degli Studi di Milano, Italy

Gabriele Bavota, University of Salerno, Italy

Grigorios N. Beligiannis, University of Western Greece, Greece

Noureddine Belkhatir, University of Grenoble, France

Imen Ben Lahmar, Institut Telecom SudParis, France

Jorge Bernardino, ISEC - Institute Polytechnic of Coimbra, Portugal

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

Ateet Bhalla, NRI Institute of Information Science and Technology, Bhopal, India

Ling Bian, University at Buffalo, USA

Kenneth Duncan Boness, University of Reading, England

Pierre Borne, Ecole Centrale de Lille, France

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

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

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

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

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

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

CONTENTS

pages 1 - 14

Combining Explicitness and Classifying Performance via MIDOVA Lossless Representation for Qualitative Datasets

Martine Cadot, Université de Nancy/LORIA, France

Alain Lelu, Université de Franche-Comté/LASELDI, France

pages 15 - 26

Testing of an automatically generated compiler, Review of retargetable testing system

Ludek Dolihal, Faculty of Information Technology, BUT, Czech Republic

Tomas Hruska, Faculty of Information Technology, BUT, Czech Republic

Karel Masarik, Faculty of Information Technology, BUT, Czech Republic

pages 27 - 35

Compiler-based Differentiation of Higher-Order Numerical Simulation Codes using Interprocedural Checkpointing

Michel Schanen, LuFG Informatik 12: Software and Tools for Computational Engineering, RWTH Aachen University, Germany

Michael Förster, LuFG Informatik 12: Software and Tools for Computational Engineering, RWTH Aachen University, Germany

Boris Gendler, LuFG Informatik 12: Software and Tools for Computational Engineering, RWTH Aachen University, Germany

Uwe Naumann, LuFG Informatik 12: Software and Tools for Computational Engineering, RWTH Aachen University, Germany

pages 36 - 52

A Programming Paradigm based on Agent-Oriented Abstractions

Alessandro Ricci, University of Bologna, Italy

Andrea Santi, University of Bologna, Italy

pages 53 - 64

Design by Contract for Web Services: Architecture, Guidelines, and Mappings

Bernhard Hollunder, Furtwangen University of Applied Sciences, Germany

Matthias Herrmann, Furtwangen University of Applied Sciences, Germany

Andreas Hülzenbecher, Furtwangen University of Applied Sciences, Germany

pages 65 - 75

An Implementation Approach for Inter-Cloud Service Combination

Jie Tao, Karlsruhe Institute of Technology, Germany

Daniel Franz, Karlsruhe Institute of Technology, Germany

Holger Marten, Karlsruhe Institute of Technology, Germany

Achim Streit, Karlsruhe Institute of Technology, Germany

pages 76 - 90

User-driven Service Retrieval Platform for Converged Environments

Edgar Camilo Pedraza Alarcon, University of Cauca, Colombia
Julian Andres Zuñiga Gallego, University of Cauca, Colombia
Luis Javier Suarez Meza, University of Cauca, Colombia
Juan Carlos Corrales, University of Cauca, Colombia

pages 91 - 109

Ad hoc Iteration and Re-execution of Activities in Workflows

Mirko Sonntag, Institute of Architecture of Application Systems, University of Stuttgart, Germany
Dimka Karastoyanova, Institute of Architecture of Application Systems, University of Stuttgart, Germany

pages 110 - 120

Modeling Disjunctive Context in Access Control

Narhimene Boustia, Computer Science Department- Saad Dahlab University of Blida, Algeria
Aicha Mokhtari, Computer Science Department - USTHB, Algeria

pages 121 - 130

Seeing the Big Picture: Influence of Global Factors on Local Decisions

Terry Bossomaier, Charles Sturt University, Australia
Michael Harré, University of Sydney, Australia
Vaenthan Thiruvardhelvan, Charles Sturt University, Australia

pages 131 - 145

An Adaptive Computational Intelligence Algorithm for Simulation-driven Optimization Problems

Yoel Tenne, Formerly Kyoto University, Japan
Kazhuhiro Izui, Kyoto University, Japan
Shinji Nishiwaki, Kyoto University, Japan

Combining explicitness and classifying performance via MIDOVA lossless representation for qualitative datasets

Martine Cadot
 Université de Nancy/LORIA,
 Département Informatique
 Nancy - France
 Martine.Cadot@loria.fr

Alain Lelu
 Université de Franche-Comté/LASELDI
 LORIA
 Nancy - France
 Alain.Lelu@univ-fcomte.fr

Abstract—Basically, MIDOVA (Multidimensional Interaction Differential of Variability) lists the relevant combinations of K boolean variables in a datatable, giving rise to an appropriate expansion of the original set of variables, and well-fitted to a number of data mining tasks. MIDOVA takes into account the presence as well as the absence of items. The building of level- k itemsets starting from level- $k-1$ ones relies on the concept of *residue*, which entails the potential of an itemset to create higher-order non-trivial associations – unlike Apriori method, bound to count the sole presence of itemsets and exposed to the combinatorial explosion. We assess the value of our representation by presenting an application to three well-known classification tasks: the resulting success proves that our objective of extracting the relevant interactions hidden in the data, and only these ones, has been hit.

Keywords-symbolic discrimination; variable interaction; machine learning; classification; non-linear discrimination; user comprehensibility; feature construction; feature selection; itemset extraction

I. INTRODUCTION

We introduce here a novel representation of an observation \times variable datatable with binary modalities. This representation aims at enlightening the interactions between variables hidden in the data. It takes into account the negative as well as positive modalities of the variables, and eliminates redundancy, since it lists the only itemsets necessary and sufficient for reconstituting the data.

In order to assess the validity and usability of such a representation, we present an application of it to classification tasks on a few well-known test datasets. It must be clear that the main subject of this paper is *not* another classification method, which should be systematically assessed against the best existing ones on a broad variety of datasets: our success on a small number of classification tasks is nothing but a sufficient clue for arguing that the proposed representation is useful in many data mining applications, whether supervised (e.g., classification) or not (e.g., data-driven modeling).

We depart here from the two main research lines in data mining:

- the statistical line relies on correlations, covariances or contingency tables for assessing the two-by-two relations between variables, and generally ignores the interactions of rank three or more.

- the symbolic line (*knowledge discovery*) is bound to extract *frequent* itemsets or association rules in sparse datatables, which restrains it in practice to enlighten the sole interactions between positive modalities of the variables, and results in an excessive accumulation of logical conjunctions needing empirical frequency thresholds.

In contrast, our method 1) takes into account the 2^N facets of each order- N relation, including negative modalities when necessary, 2) self-limits the number and nature of the extracted relations to the ones necessary and sufficient for reconstituting the whole datatable, up to a permutation of the observations.

The assessment of a new data representation is far from being trivial: it is widely agreed that unsupervised data representations are difficult to assess. In our case an extra difficulty lies in the fact that few people use and interpret order-3 and higher interactions discovered in an unsupervised framework. Using a supervised one is thus a way for us to circumvent this limitation. This is why we have decided to feed a classic and well-known machine learning method (*Naïve Bayes*) with extra data issued from our MIDOVA representation of the datatable as is now exposed.

The illustrative objective of this paper is to present a proof-of-concept solution for a problem of ever-growing importance for software industry: in the framework of discrimination tasks, when facing massive and mostly qualitative data as it becomes common now, there is a growing need to explicit the reasons underlying a given discrimination, i.e., to uncover the variables or combination of variables intervening in the discrimination function.

The core principle of Support Vector Machines is that classification performance is increased by *increasing the dimensionality of the representation space* in which the discrimination task has to be performed, and not by reducing it. This paradox is explained by the concept of interaction: two (or more) variables may not have the same effect as each one separately. In particular the combination of two (or more) variables may impact the target variable, while none of these variables would do that individually. This principle inspired us for setting up an enlarged data-space in classification problems involving binary variables, for the time being: our aim is to detect *all* the order- k interactions, so as to select those involving the target variable. We will show below that the number of combinations to consider is tractable: 1) this number is intrinsically limited by the

number of described entities (or “cases”, “individuals”, “observations”) in the data, 2) at the k -th level, it is far below the number of theoretical combinations – as k increases, it decreases abruptly after its initial growth phase. Our solution has been briefly exposed yet in DBKDA 2010 [1]. As subtle combinatorics considerations are to be developed, we will take more pages here, and thoroughly examine all the facets of our solution considering a single toy example. Let us turn now to our main topic.

For supervised classification tasks, the main criterion is the overall classification performance, i.e., best generalizing power. In this regard, it is widely admitted that kernel Support Vector Machines [2] have outperformed their competitors. But in many application areas, the explicitness of the discrimination function is also a major criterion: kernel SVMs are blackboxes not akin to “explain” their decisions. The “kernel trick” is a powerful by-pass for computation costs, but at the expense of turning the decision process blind. This is a big concern in domains with life-threatening issues, such as medical or military ones. It is impossible to let a machine take (or suggest) such decisions of vital importance without explaining why, i.e., without clarifying what variables or combination of variables are involved in each specific decision-making.

In this respect, many other classification methods, whether linear as Naïve Bayes [3] or Fisher discrimination [2], or non-linear, such as Rule-based Classifiers [4] or Learning Classifier Systems [5], are *explicit*: they display (or may display) the (possibly weighted) list of variables or combinations of them leading to their classification decision.

Our objective is to meet both criteria of explicitness and performance, restricted here to the case of Boolean variables (i.e., qualitative variables with two modalities, True and False, noted by a and \bar{a} for the variable a). The solution we propose consists of expanding the original variables with Boolean conjunctions of these ones. This idea has yet been worked out [6] in the framework of the Apriori method for extracting frequent itemsets [7]. Let us recall that an “itemset” as defined in Apriori is an unordered list of variables, its “support” is the number of co-occurrences of these variables in the dataset, and if the support exceeds a given threshold, the itemset is said “frequent”. In this paper, we will deepen the presentation of our MIDOVA method, first described in French in [8], then in English in DBKDA 2010 [1]. As it is an unsupervised information extraction method, we will assess its performance applying it to supervised machine learning tasks, and comparing its results to the best published ones. The excellent classification performance obtained is mainly due to the conjunction of its original features, among which: it takes into account negative modalities as well as positive ones, it replaces the straightforward support criterion of Apriori-like methods by our “residue” criterion, detailed below.

First we will present the MIDOVA representation of a datatable: its context and motivation, its general and core principles, and an overview of the algorithm and its pseudo-code expression. Then we will detail the whole MIDOVA process on a toy dataset. At last the experimental section will present the application of MIDOVA expansion to the most

basic discrimination problem: 2 classes and qualitative data, taken out of the UCI repository. Conclusions will be drawn, as well as perspectives.

II. MIDOVA REPRESENTATION

After some generalities on MIDOVA, the reader will be presented a small example illustrating the essential concepts of component of an itemset, degree of freedom, support, residue and gain. We will insist on the concept of variation interval and illustrate it by Venn diagrams. We will confirm that MIDOVA uncovers the two clusters we had placed in our dataset. At the end we will develop some quantitative assessments of MIDOVA, especially in comparison to Apriori.

A. Context of the MIDOVA representation

We consider the case of a relation R between two sets, a set S of n individuals (or instances) s_i ($1 \leq i \leq n$), and a set V of p binary variables (or variables) v_j ($1 \leq j \leq p$). For example the individuals are patients in a hospital, and the (dichotomous) variables are the symptoms they experience, or not. Or the individuals are clients of a supermarket, and the variables are the products they are akin to buy, or not.

A first way to represent these data is as a list of lists: the lists of symptoms experienced by each patient, or the lists of commodities bought by each client, i.e., a set of sales slips. The second representation is more appropriate for reasoning: a Boolean matrix with n rows and p columns, and value 1 at the crossing of row i and column j if the individual experiences the symptom, or else the value 0. It is well known that these two representations are equivalent, the first one being a generally compact form of the second (as it gets rid of the zero values). The datasets illustrating our method are formatted as Boolean matrices.

We will propose a third representation, which is a set of itemsets. This representation is rooted in the extension of the statistical concept of contingency table between two variables a and b , to the concept of “contingency hyper-table” between more than two variables. The classic contingency table includes values in the four cells describing how the total number n of individuals distributes respectively along the four crossings of a and b (i.e., the counts of individuals who simultaneously satisfy a and b , a and \bar{b} , \bar{a} and b , and \bar{a} and \bar{b}). Note that the first count is the support of itemset ab . In the same way, the three-way contingency table (between three variables) will be a cube with eight cells, and the appending of one more variable doubles the number of cells where the individuals distribute. As the sum of these cells is n , their content is smaller and smaller, and empty cells are more and more common. We call *components* of our k -itemsets (itemsets of length k , i.e with k variables) the cells of the k -way contingency table. Note that our definition of an itemset is more general than the one generally accepted, in that it includes all the cells of the contingency table, and not the sole “True, True, ...True” one. Note also that we have started from the 2-way contingency table, which is the most common one, by increasing the dimension, but for the sake of generality we also define the 1-way contingency tables for the 1-itemsets, i.e., with only one

variable and the empty itemset of 2^V , which we will admit being satisfied for all the individuals.

B. Principles for building the MIDOVA representation

The core principle of MIDOVA is based on the properties of contingency tables, which can be extended to hyper-tables. They are expressed in terms of marginal totals and degrees of freedom.

1) Marginal totals and degrees of freedom

The sums of all the counts corresponding to a variable, e.g., a, in a k-way contingency table are the exact counts of the (k-1)-way contingency tables with all variables except a. For example, in Figure 2, Row 2, the first 2-way contingency table corresponds to variables a and b, with four counts (3 individuals who verify a and b, 4 who verify a but not b, 2 who verify b and not a, 6 who verify nor a nor b). In its right-hand column, marginal totals are the two counts of a (7 individuals who verify a, 8 who do not verify a), the same as those written in the 1-way table of a (Figure 2, Row 1). In the bottom row of this 2-way table, the marginal totals are the two counts of b (5, 10) also written in the 1-way table b. In other words, the marginal sums are forcing the counts in the cells of the table, restraining the “freedom” of their contents.

The concept of “degrees of freedom” embeds the number of cells whose content may be fixed independently from the others – within limits we will express below. In our case of Boolean variables, the degree of freedom of each hyper-table is equal to one because all counts can be written as an algebraic expression of x (where x is the count of some fixed cell), and of the marginal values (see Figure 1 and Figure 2, and details of calculations in Section C).

2) MIDOVA indicators

In the previous paragraph, we have seen that the values of the cells of a K-way contingency table (i.e., the components of a K-itemset), are all linked to the value x of a single cell (i.e., to a single component) and to marginal values (i.e., to components of its sub-itemsets) by means of simple algebraic expressions. At step k-1, the x value is unknown, but it is possible to obtain its *variation interval*, i.e., its different values. For example, for itemset ab in Figure 1, the variation interval of x is [2 ; 7], and for itemset abc in Figure 2, the variation interval of x is [1 ; 2] (for details of calculations, see Section C; for an interpretation of the variation interval see Section F 4).

Three useful properties of itemsets are ensuing (their proofs are in [8].)

- The variation intervals of the all components of a k-itemset are entirely determined by the components of its (k-1)-itemsets.
- The components of a k-itemset all have the same amplitude noted L, “liberty”.
- L is a non-increasing function of k.

We define two parameters: 1) Mr, the “MIDOVA-residue”, equal to $2^{k-1}e$, where e is the gap (i.e., absolute difference) between the support and the closest bound of its

variation interval, 2) Mg, the “MIDOVA-gain”, which is proportional to the difference between the support s and the center c of its variation interval, $Mg=2^{k-1}(s-c)$. Let us recall that the support is the content of the “True True...True” cell corresponding to the case of values of variables all equal to 1.

Mr is a non-negative integer, which cannot exceed the value n/2 where n is the total number of subjects. When the residue Mr of a k-itemset is zero, it remains no more liberties (L=0) for the k'-itemsets (where k'>k) including the same variables. This frozen situation stops their computation. This k-itemset is interesting in that it corresponds to an exact relation between all its variables. Its sub-itemsets may be also interesting: they correspond to relations between fewer variables, thus more general, but these relations are not exact, and have a number of counter-examples which increases with Mr.

Mg is an integer that takes values between -n/2 and n/2. If the gain Mg of a k-itemset is zero, the relation between the k variables can be rigorously deduced from the lower-level relations between k-1 variables. In the opposite case, the greater the absolute value of Mg, the larger the unexpectedness in this relation, and the more interesting it is. If its value is positive, the appended variable increases the relation between the previous variables, and decreases it otherwise.

C. Illustration of the MIDOVA principles

At the left of Figure 1, we have represented the 0-way contingency table corresponding to the 0-itemset, with 0 variable, always true, and the 1-way contingency tables respectively corresponding to the 1-itemsets a (true for 7 subjects) and b (true for 5 subjects), with the total n in the marginal cell. The 2-way contingency table corresponding to the 2-itemset ab is displayed at the right-hand part of the Figure 1. The components of itemset a are in the marginal column and the component of itemset b in the marginal row, and among the four cells of the table, only one cell can be affected independently of the other cells. We have chosen the cell corresponding of the number of individuals who satisfy a but not b, and the unknown number x is written in this cell.

n
15

a	\bar{a}	Tot.
7	8	15

b	\bar{b}	Tot.
5	10	15

	b	\bar{b}	Tot.
a	7-x	x	7
\bar{a}	8-(10-x)	10-x	8
Tot.	5	10	15

Figure 1. Expression of the 4 components of itemset ab given a, b and the total n

The other counts of the table are relative to x and the four marginal sums, their algebraic expression can be easily derived. For example, as the sum of the two cells in the first line is 7 (in blue, as the sum of the two cells in the first column of the 1-way table of 1-itemset a, which indicates that 7 subjects verify a), the number of subjects who verify a and b is 7-x. In the same way, as Column 2 contains two cells, which total is 10 (in red, in the marginal row of the 2-way table, and in the second column of the 1-way table of 1-itemset b), and as the cell in the first

row contains x , the second cell contains $10-x$. And the value of the last cell derives by difference between the total of Row 2 (8, bold and blue) and the content ($10-x$) of the other cell of Row 2.

n	a	\bar{a}	Tot.	b	\bar{b}	Tot.	c	\bar{c}	Tot.
15	7	8	15	5	10	15	7	8	15

	b	\bar{b}	Tot.		c	\bar{c}	Tot.		c	\bar{c}	Tot.
a	3	4	7	a	5	2	7	b	2	3	5
\bar{a}	2	6	8	\bar{a}	2	6	8	\bar{b}	5	5	10
Tot.	5	10	15	Tot.	7	8	15	Tot.	7	8	15

		b			\bar{b}		Tot.			c		\bar{c}		Tot.
		c	\bar{c}	Tot.		c	\bar{c}	Tot.		c	\bar{c}	Tot.		
a	3-x	x	3	4-(2-x)	2-x	4	5	2						
\bar{a}	2-(3-x)	3-x	2	6-(5-(2-x))	5-(2-x)	6	2	6						
Tot.	2	3		5	5									

Figure 2. Expression of the 8 components of itemset abc given ab, ac, bc, a, b, c and n ($a=v_2, b=v_4, c=v_3$ from Table 1)

The value of x can vary between 2 and 7 because the four counts of the 2-way table ($7-x, x, x-2$ and $10-x$) must be non negative. It follows that the liberty of the ab itemset is $L=7-2=5$. In Figure 2, we have the same items a and b, the ab itemset has been fixed ($x=4$), and a new item c has been added, with ac and bc itemsets, which are all known through their tables. The abc itemset is unknown, but the eight counts of the 3-way table in the third row depend on the value of x and on the marginal sums. It may be observed that the value of x is between 1 (as $x-1$, which is the count of individuals satisfying b and c but not a, cannot be negative) and 2 (as $2-x$, which is the count of individuals satisfying a but neither b nor c, may not be negative), and so is $L=2-1=1$ for the abc itemset. For the sequence of itemsets a, ab, abc, the corresponding sequence of the L values of liberty is 15, 5, 1.

The computation of M_r and M_g for the abc itemset is developed in Section F.4: in the case of $x=1$, the three variables a, b, c are equivalent to the variables v_2, v_4, v_3 of the Table I

D. Representation of a MIDOVA sequence

When one knows the count of a unique cell per each contingency table (there are 2^p such tables), it is enough for reconstructing the whole relation R. To this special role we will assign by convention the cell where all variables are set to 1, corresponding to the support of the itemset. For example in Figure 2 the relation between a, b, c is wholly defined, if $x=1$, by: $\emptyset(15), a(7), b(5), c(7), ab(3), ac(5), bc(2), abc(2)$.

Generally (and fortunately), not all itemsets appear in the MIDOVA representation. As soon as an itemset is frozen ($M_r=0$), no following itemset is set up (i.e if abc is frozen, no abcd, abce, abcde, ..., itemset will ensue). When the grand total n distributes among the 2^k cells of a k-way contingency table and $n < 2^k$, then one cell at least is empty, and no more-than-k-dimensional table will be created. The maximum order of a component of R will be $\log_2(n)+1$.

E. MIDOVA algorithm

The MIDOVA algorithm is a levelwise algorithm, derived from the Apriori one [7]. The major difference between our algorithm and the Apriori one is our criterion of positive M_r allowing to enlarge a k-itemset into a k+1-itemset, instead of a support exceeding a threshold. In our algorithm, we have added (through the function "conditions") the possibility of saving the only itemsets with a support, and/or a gain and/or a residue greater than given thresholds. The description of MIDOVA algorithm comprises 3 parts: the variables, the functions and the main procedure.

1) Variables

- V : list of the variable headings in lexicographic order
- M : boolean matrix of the relation R
- L_0 : empty list
- e : element of a list
- $it1$: k-association, i.e., list of k variable headings in lexicographic order with their 2^k components (number of individuals in each cell)
- $it2$: the same as $it1$, with length k+1
- $it0$: empty association
- k : length of the associations at the current step
- I : global list of the Midova-oriented representation, accompanied for each association with its heading and other measures computed from its components
- $L1$: list of associations built at step k
- $L2$: sub-list of associations saved for the k+1 step
- M_r : Midova residue
- $Possible_follow_up$ is a boolean variable, true if it is possible to extract in $L2$ an association built upon association $it1$

2) Functions

conditions(it): boolean function; true if association it checks up the conditions specified by the user, for example thresholds of support, gain and/or residue, computed starting from the components of the association.

append(I, it) adds the it association to I, keeping just the heading and the sole part of information issued from the components wanted by the application designer, e.g., support, gain, residue.

parc_L2(it, k, j) boolean function; true if the heading of the j^{th} association in the $L2$ list next to it has the same (k-1) first elements as the it heading.

create_it(M, i) builds an association starting from its heading and its list of components computed from M.

succ(L2, it1, j) identifies the successor of $it1$ located j slots further in the $L2$ list.

extract(it2) extracts the (k-1)-subassociations in the heading of $it2$ except the two first; e.g.: **extract("abdf")** yields the list ["adf", "bdf"]

rech_co_L2(M, it1, k, j) tests the grouping of the k- association $it1$ with the k-association $it3$ positioned j slots further in the $L2$ list. It yields the $it2$ heading by appending the last variable of $it3$ heading to the list of variables of the $it1$ heading. It checks then whether any sub-association included in $it2$ exists in $L2$. If not, the computed association is void. Else it creates the $it2$ association appending to its heading the values of its 2^k components computed from M.

Pseudo-code for rech_co_L2 function

```

rech_co_L2(M, it1, k, j)
it3=succ(L2, it1, j)
it2=concat(intit(it2),intit(it3)[k])
succeed=True
list_it←extract(it2)
it3=succ(L2, it3,1)
for it in liste_it do
    while not(it3=it0) and (it3<it) do
        it3=succ(L2, it3,1)
    end
    if not(it=it4) then
        succeed=False
        return it0
    end
end
it2←create_it(M, it2)
return it2
    
```

3) *Main procedure*

Pseudo-code for generating the MIDOVA representation

```

#Initialisation :
K ← 1 ; I ← L0 ; L1 ← L0 ; L2 ← L0
for e in V do
    it ← create_it(M, e)
    L1 ← L1 + it
end
for e in L1 do
    If conditions(e) then append(I, e) end
    If Mr(e)>0 then L2 ← L2 + e end
end

#further steps
while not (L2 = L0) do
    k ← k+1, L1 ← L0
    for it1 in L2 do
        j←1
        Possible_follow_up ← parc_L2(it1, k, j)
        while Possible_follow_up do
            it2 ←rech_co_L2(M, it1, k, j)
            if not(it2=it0) then L1 ← L1 + it2 End
            j←j+1
            Possible_follow_up ← parc_L2(it1, k, j)
        end
    end
    L2 ← L0
end
for e in L1 do
    If conditions(e) then append (I, e) end
    If Mr(e)>0 then L2 ← L2 + e end
end
end
    
```

F. *Running MIDOVA on a toy dataset*

We illustrate here the whole MIDOVA operation line on a small example of dataset, extending from the raw matrix representation, to the interpretation of the final MIDOVA results. For the sake of clarity, we take a simpler example than the partial one presented in the above subsection. Table 1 displays our artificial dataset, showing the Boolean values of 15 subjects s1, s2, ..., s15 for 10 variables v1, v2, ..., v10. One may read, for example, that for the subject s2, 5 variables upon 10, i.e., v1, v2, v3, v4 and v9, are true; or that the variable v6 is true for 4 subjects upon 15, i.e., s10, s12, s14 and s15.

TABLE I. BOOLEAN TABLE OF 15 SUBJECTS (E.G., PATIENTS) AND 10 VARIABLES (E.G., SYMPTOMS)

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10
s1	1	1	1	1	1	0	0	0	0	0
s2	1	1	1	1	0	0	0	0	1	0
s3	1	1	1	0	1	0	0	0	0	0
s4	1	1	0	1	1	0	0	0	0	0
s5	1	1	0	0	1	0	1	0	0	0
s6	1	1	1	0	0	0	0	0	0	0
s7	1	0	1	0	1	0	0	0	0	0
s8	1	1	1	0	0	0	0	1	0	0
s9	1	0	0	1	1	0	0	0	0	0
s10	1	0	0	0	0	1	1	0	0	0
s11	1	0	1	0	0	0	0	1	1	0
s12	1	0	0	0	0	1	1	0	0	0
s13	1	0	0	1	0	0	1	1	1	0
s14	1	0	0	0	0	1	1	0	1	0
s15	1	0	0	0	0	1	1	1	1	0

1) *Scrutinizing a few 2-itemsets*

The link between two variables is akin to be more or less pronounced, spanning from a complete opposition to a complete similarity through a complete unrelatedness. We exemplify below these three situations of respectively: 1) opposition, or contradiction, 2) linkage, or attraction 3) independence, or lack of connectedness.

The itemset $A = \{v4 ; v6\}$

- includes two variables, thus its *length* k is 2.
- As the variable v4 is true for the five subjects s1, s2, s4, s9 and s13, it appears that no subject satisfying v6 belongs to this list, thus the *support* of A is zero.
- Its support could have been 1, 2, 3 or 4, but not further, depending on the possible number or common subjects. The *variation interval* of the support of A is therefore [0 ; 4], with a lower bound $b_{inf}=0$ and an upper bound $b_{sup}=4$.
- As seen above, its *residue* is a function of the gap (absolute value of the difference) between its actual support and the closest bound; in this case, $Mr=2^{k-1}(s-b_{inf})$ and its value is zero since $s=b_{inf}=0$.
- Its *gain* is a function of the difference between its actual support and the center of the variation interval ($c=2$), $Mg=2^{k-1}(s-c)$, i.e., -4.

This itemset is interesting in that it sheds light on an opposition between v4 and v6: the subjects who satisfy v4 do not satisfy v6, and vice-versa. This is precisely the type of knowledge we try to mine out of the data. But its super-itemsets $\{v1, v4, v6\}$, $\{v2, v4, v6\}$, ..., $\{v10, v4, v6\}$ are uninteresting: no need to return to Table 1 to conclude that their supports are zero, this conclusion proceeds from the zero support of A. We call A a *frozen itemset* for it does not generate super-itemsets carrying an extra knowledge, out of its own contribution. The zero value of the residue points at this frozen situation.

The itemset $B = \{v6, v7\}$ is true for 4 subjects, those who satisfy v6; v7 is true for these 4 subjects and two other ones, s5 and s13. Its support is 4 and its variation interval is [0; 4] as above. As the support equals the lower bound, the residue

Mr of B appears to be zero and the gain Mg is 4. B bears interesting information, i.e., all the subjects for which v6 is true satisfy also v7. This can be also expressed by the *association rule* $v6 \rightarrow v7$ whose *confidence* is 100%. Its super-itemsets are uninteresting: no need to return to Table 1 for delineating the subjects who satisfy the 3-itemset {v6; v7; v9}, they are exactly those satisfying {v6; v9}. The itemset B is frozen, as pointed out by its zero residue.

The itemset $C = \{v6, v9\}$ is true for 2 subjects, s14 and s15, sharing both variables, while v6 and v9 are respectively true for 4 and 5 subjects. The variation interval of this itemset is again [0; 4], but the support $s=2$ of C is now central in the interval, which corresponds to a residue of value 4 and a gain of 0. This zero gain shows that the relation between v6 and v9 is uninteresting. Conversely, as Mr is greater than zero, this itemset is not frozen, and one has to consider its derived and possibly interesting super-itemsets.

2) *MIDOVA selection of the k-itemsets*

The MIDOVA algorithm works level-wise, which means that once established, the level-k itemsets are combined for deriving the level-(k+1) ones. A level-(k+1) itemset needs k+1 level-k itemsets. In this way the 3-itemset {v2, v3, v4} derives from the three 2-itemsets {v2, v3}, {v2, v4} and {v3, v4}.

- Level 1: the 1-itemsets, made of a single variable, are generated ; there are ten of them.
- Level 2: the 1-itemsets are combined by twos for creating the 2-itemsets. For generating the sole knowledge-carrying 2-itemsets, the 1-itemsets with non-zero residues are selected, which eliminates the two variables v1, true for all the subjects, and v10, true for none of them (their residues are zero, and their respective gains are 7.5 and -7.5). In this way 28 2-itemsets are to be processed instead of the 45 ones when keeping v1 and v10. Considering these 28 itemsets, 8 are frozen ($Mr=0$), among which 7 have zero support and gains between -6 and -4, and one has support and gain values of 4. They won't contribute to build the upper levels, but as their gain values are important, they are left apart for the final interpretation step. The 20 remaining 2-itemsets are kept for building the next levels. Four of them have a zero gain, the others spread from -4 to 3.
- Level 3: The 20 2-itemsets with $Mr \neq 0$ are combined by threes for building the 3-itemsets, yielding 22 of them, among which only three have a non-zero residue; therefore, this sole number prevents building any 4-itemsets. Their gain values are zero. Among the remaining 19 with a zero residue, two only have a non-zero gain, i.e., {v2, v3, v4} with a support of 2 and a gain value of 2, and {v2, v3, v5} with a support of 2 and a gain value of -2.

In this way we have built the wholeness of the k-itemsets akin to provide pieces of information about the dataset. Throughout three steps, 10, 28 and 22 itemsets respectively have been considered, summing up to 60 (see Annex).

Among these ones, those taken into account for building the ones at the next k+1 level amounts to respectively 8, 20 and 3; those with non-zero gain have been 10, 24 and 2, establishing in this way a total of 26 interesting relations between variables (out of the trivial 1-itemsets).

3) *Differences between MIDOVA algorithm and Apriori-like ones*

In Table II, the wholeness of the 1023 k-itemsets ($k > 0$) potentially built starting from the data are split up and counted in reference to the zero value, or not, of their residues and gains. They yield from the MIDOVA process parameterized without any residue threshold instead of the threshold value 1 assigned above. There are then as many k-itemsets as combinations of variables, i.e., $1024 = 2^{10}$. The first itemset is the void one, which includes no variable, and is true for 15 subjects – it has been taken out from the itemset list as a trivial and uninteresting one. The last one is the « full » itemset which includes all the variables, but is true for no subject. The 26 interesting itemsets are emphasized in boldface, so as to point out the efficiency of our algorithm compared to a brute force one who would examine the 1023 itemsets. It is noticeable that, in the scope of a fair comparison between the Apriori algorithm and ours, Apriori should be considered as such: its efficiency follows from the sole principle of applying a threshold to the supports, an option we have discarded in order to retain the itemsets with zero supports, as these ones mostly points to interesting opposition relations. Moreover, 7 itemsets out of the selected 26 ones have a zero support.

TABLE II. HOW THE 1023 ITEMSETS EXHAUSTIVELY ISSUED FROM TABLE I DISTRIBUTE AS REGARDS TO THE VALUE ZERO OR NOT OF THEIR MR AND MG INDICES.

		Number of k-itemsets										
	k	1	2	3	4	5	6	7	8	9	10	Total
Mr=0	Mg=0	17	115	210	252	210	120	45	10	1		980
	Mg≠0	2	8	2								12
Total		2	25	117	210	252	210	120	45	10	1	992
Mr>0	Mg=0		4	3								7
	Mg≠0	8	16									24
Total		8	20	3								31
Total		10	45	120	210	252	210	120	45	10	1	1023

Going on with the subject line of enlightening the differences between our principles and Apriori's, we have shown in Figures 3 and 4 how do interesting itemsets rise or not in both methods, starting from the exhaustive 1023 itemsets:

- The 623 firsts, lexically ranked for each increasing value of k ($1 < k < 6$): 45 2-itemsets, 120 3-itemsets, 210 4-itemsets, 252 5-itemsets).
- The remaining ones ($k \geq 6$) have zero-valued supports, residues and gains.

In both figures, the x axis indicates the rank of the itemset according to the above-mentioned ordering, and the dotted line visually recalls the number k.

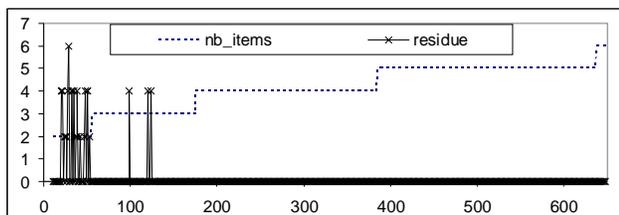


Figure 3. Residues of all k-itemsets of Table I.

In Figure 3, the y axis specifically stands for the MIDOVA residues, in order to enlighten how soon our algorithm locates the « no-future » value of k, thus how soon the algorithm is stopped.

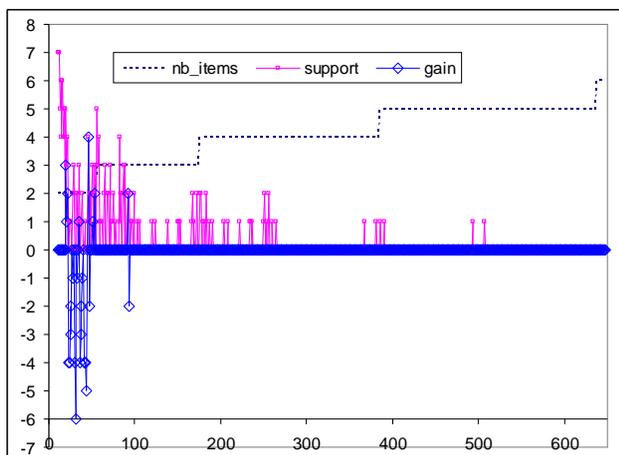


Figure 4. Interesting k-itemsets of Table 1, mined with MIDOVA or Apriori (with threshold 0)

In Figure 4, the y axis stands for the support and gains of the successive itemsets. It visually jumps out that 1) gain and support measure totally different phenomena, 2) MIDOVA does detect the strong gain values that clearly stands out as grouped for the small values of k. These two features explain both the good performance of MIDOVA, able to detect the wholeness of the interesting itemsets, and only these ones, including those with zero support, and its relative efficiency, compared to Apriori parameterized with a zero support threshold.

More precisely, one may read in Table III that among the 627-528=99 k-itemsets extracted by MIDOVA or Apriori with zero threshold, only 19 of them (17 2-itemsets and 2 3-itemsets) are simultaneously selected by the two methods – and the number of common items would still decrease using Apriori with a threshold. This confirms that the 26 inter-variable relations mined out from the Table I are different by nature from those issued from the Apriori family algorithms.

TABLE III. HOW THE 627 K-ITEMSETS (1<k<6) ISSUED FROM TABLE I DISTRIBUTE, ACCORDING TO ZERO OR NON-ZERO VALUES OF SUPPORTS AND GAINS

supp>s	gain >g0	2	3	4	5	Total
support=0	Mg=0	9	82	189	248	528
	Mg≠0	7				7
support≠0	Mg=0	12	36	21	4	73
	Mg≠0	17	2			19
Total		45	120	210	252	627

As we have already interpreted the meaning of three examples of 2-itemsets, we now interpret the only two interesting relations between 3 variables.

4) Interpreting the 3-itemsets with non-zero gain

Let us examine first the itemset $D=\{v2, v3, v4\}$. In Figure 5 a Venn diagram shows how the 15 subjects distribute among the three variables $v2, v3$ and $v4$. In this ensemblist layout, the subjects are splitted in 8 parts according to their values of the three considered variables. For example $s7$ and $s11$ are in the segment of $v3$ exterior to $v2$ and $v4$, as their values for $v3$ are 1, whereas they are zero for $v2$ and $v4$. In the same way the $s1$ and $s2$ subjects lie in the central part common to the three variables, as their values are 1 for all of them. Thus the support of the itemset D is 2.

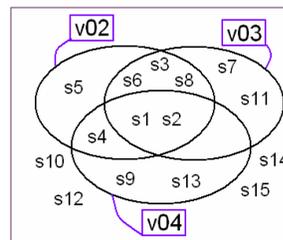


Figure 5. Venn diagram of the {v2, v3, v4} itemset.

The variation interval of the support can be found by trying to modify the support of D without modifying the supports of its component 2-itemsets. This can be done by moving the subjects from one area to another one: the only possible configuration is shown in Figure 6.

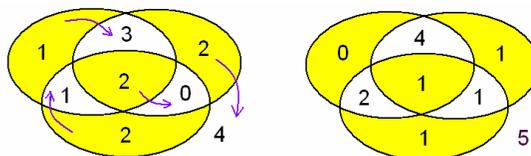


Figure 6. Transfer of 4 subjects of the {v2, v3, v4} itemset: the movements are drawn in the left-hand figure, resulting in the right-hand configuration.

In the left part of Figure 6, blue arrows indicate the authorized movements of the subjects (starting from a yellow area, where the number of negative variables is even, to a white one). As far as each support of the 2- and 1-itemsets

includes as much yellow areas as white ones (one each for the 2-itemsets and two for the 1-itemsets), their values are kept unchanged. For example the sub-itemset {v2, v3}, which included at the left a white area with 3 subjects and a yellow one with 2, now includes at the right a white area of 4 and a yellow one of 1, while its support stays the same ($5=3+2=4+1$) – obeying a kind of conservation principle, so to speak. The sub-itemset {v2}, which included two white areas with 1 and 3 subjects, and two yellow ones with 1 and 2 subjects, now includes two white ones with 2 and 4, and two yellow ones with with 0 and 1, whereas its support keeps constant ($7=(1+3)+(1+2)=(2+4)+(0+1)$). Implementing these changes of the variation interval can be done by just modifying the four values in the datatable. This can be done in different ways, one of which is displayed in Figure 7. In the left-hand part of the table the values of Figure 1 are reproduced, and the values to be changed for decreasing the support of D from 2 to 1 are highlighted in yellow; at the right-hand part of the table, these values have been modified.

Figure 6 shows the only possibility for the variation of the D itemset subjected to the stability conditions of the supports of its sub-itemsets. It follows that the variation interval of its support is [1; 2], of center 1.5 and gain $Mg=2^{k-1}(s-c)=2^2(2-1.5)=2$, which points out a tight relation between v2, v3 and v4, relatively to the three underlying 2-relations, i.e., (v2 and v3, v2 and v4, v3 and v4).

	v02	v03	v04	v02	v03	v04
s01	1	1	1	0	1	1
s02	1	1	1	1	1	1
s03	1	1	0	1	1	0
s04	1	0	1	1	0	1
s05	1	0	0	1	1	0
s06	1	1	0	1	1	0
s07	0	1	0	0	0	0
s08	1	1	0	1	1	0
s09	0	0	1	1	0	1
s10	0	0	0	0	0	0
s11	0	1	0	0	1	0
s12	0	0	0	0	0	0
s13	0	0	1	0	0	1
s14	0	0	0	0	0	0
s15	0	0	0	0	0	0

Figure 7. An example of modification of 4 subjects of the {v2, v3, v4} itemset for implementing the changes in Figure 6.

The second 3-itemset with non-zero gain is $E=\{v2, v3, v5\}$, which Venn diagram is shown in Figure 8.

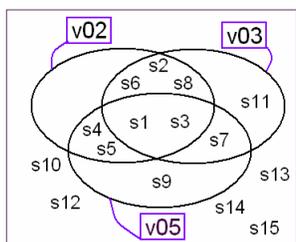


Figure 8. Venn diagram of the {v2, v3, v5} itemset.

In the same way Figure 9 shows the only way to move subjects in order to modify the support of E without modifying the supports of its component 2-itemsets, which yields a variation interval [2; 3] for the support, and thus a gain of -2 expressing a loosening of the link between the 3 variables v2, v3 and v5.

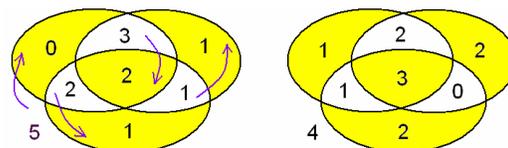


Figure 9. Transfer of 4 subjects of the {v2, v3, v5} itemset: the movements are drawn in the left-hand figure, resulting in the right-hand configuration.

5) Global interpretation

The reader may probably have observed that our example data had a special structure: apart from v1, always true, and v2, never true, the variables v2 to v5 mainly characterize the subjects s1 to s9, and so do the variables v6 to v9 for the subjects s10 to s15. This fact explains why the k-itemsets with non-zero gain extracted by MIDOVA are quasi-exclusively of order 2, because the dominant structure in the table is an opposition between two subject clusters. In this framework, the 2-itemsets that link the variables characterizing one cluster are mainly endowed with a high positive gain, whereas those linking two variables with different clusters tend to have highly negative gains, while being mostly characterized by a zero support. We can conclude that MIDOVA has uncovered the knowledge that had been incorporated in the data, i.e., the existence of two contrasting clusters, a bit blurred with some noise.

G. Performance assessment of MiDOVA vs. Apriori

TABLE IV. NUMBER OF ITEMSETS AS A FONCTION OF 1) THEIR LENGTH, 2) THE ALGORITHM (WBC DATASET, FROM UCI REPOSITORY)

k	Apriori	MIDOVA with all variables			MIDOVA with class: benign		
		Mr=0	Mr>0	Total	Mr=0	Mr>0	Total
2	4095	1652	2443	4095	23	66	89
3	121485	23750	7931	31681	1119	368	1487
4	2672670	12134	1174	13308	762	83	845
5	46504458	186	0	186	18	0	18
6	666563898	0	0	0	0	0	0
7	8,09 E+9						
...	...						
Tot.	2,48 E+27	37722	11548	49270	1922	517	2439

As an example, we have performed MIDOVA and Apriori on the UCI test-data Wisconsin Breast Cancer (WBC) [9; 10]. This test set comprises ten categorical variables, among which the target variable with the Benign /

Malignant modalities, 8 with ten modalities and 1 with 9 modalities; all these were translated into 91 dichotomous variables, observed for 699 individuals. In Table IV the length of itemsets is in the first column; in the second and third ones are displayed the number of itemsets obtained with Apriori and MIDOVA respectively, with threshold 0, separating those with zero residue, the others, and those who include the target variable with value = Benign.

The Apriori algorithm yields 4095 2-itemsets (see Table IV), which are all the possible combinations of these 91 variables 2 by 2; MIDOVA too because none of these 91 variables could be eliminated, by default of zero residue value. However more than a third of these 4095 itemsets (1652) have a zero residue, and are thus eliminated from the list of itemsets grounding the building of 3-itemsets. At next step ($k=3$) the number of itemsets grows significantly (31 681, i.e. 7.7 times more than at step 2), but much lesser than with Apriori (29.7 times more than at step 2). Among the 3-itemsets generated by MIDOVA, a large proportion has a zero residue (23 750, i.e., more than 70%). From step 3 to step 4 the number of itemsets is multiplied by 22 with Apriori, 0.42 with MIDOVA (see Figure 10). At step 5, 186 itemsets are kept by MIDOVA, all of them with a zero residue, which explains that no itemset of length >5 exists, at the same time when the number of itemsets keeps growing exponentially in Apriori. At last, 1922 itemsets have been kept as candidate variables (“expansion” of the original ones) for predicting the benign target modality.

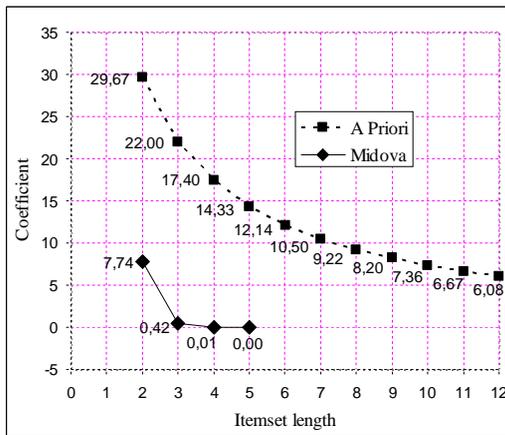


Figure 10. Multipliers for the number of itemsets – an Apriori and MIDOVA comparison (WBC dataset, from UCI repository)

In this way, only a minor part of the 49 270 components of the MIDOVA decomposition is used for the classification task, i.e., the most relevant itemsets in this respect: the ones with $M_r=0$, $M_g \neq 0$ and including the label variable C , thus expressing the tightest relationship with this variable. We will take advantage of this observation in next section III.

H. Effectiveness of MIDOVA

When no threshold support is applied, the maximum complexity at level 2 is $O(N_s, N_v^2)$, where N_s and N_v are the

number of subjects and variables respectively. The complexity at the further levels depends heavily on the presence or not of interactions, and their distribution along the successive levels. For example, a structure of simple clusters, i.e., “blocks”, mainly results in 2-itemsets, with few higher-order ones. It is the case in our toy example in subsection F, where the only two order-three itemsets may be considered as random noise.

We are aware that many enhancements of the Apriori algorithm have been published since [7]. However these variants do not change, to our knowledge, the basic principles at work, and could also be applied for increasing the efficiency of MIDOVA – this prospect is one of our current interests.

III. NAÏVE BAYES CLASSIFICATION OF THE EXPANDED DATATABLE

We address now the application of MIDOVA to the classification problem. For the sake of simplicity we will tackle the 2-class problem. The class variable C is thus a binary one, and its modality is known for each observation of the learning set. As MIDOVA gives a complete view of how any variable is related to the other ones, C included, we have applied it to the learning set and we have selected the only subset of k -itemsets involving C with 0-valued residue, and extracted their 0-valued components (corresponding to the cells of count 0 in the k -way contingency table). Each component results in a new variable, product of the values 0 or 1 of its variables (except C). We call “MIDOVA expansion” the set of components and “MIDOVA-expanded datatable” the datatable with the new variables.

For the sake of simplicity again we will use the most basic classification approach, i.e., the Naïve Bayes method. This approach poses the hypothesis of independent variables, i.e., the log-odd for a data-vector to belong to class C is the sum of the contributions from priors and separate contributions from each of the variables.

Which translates, in our specific case of two classes C and $\neg C$ and binary components of data-vectors $\mathbf{x}=\{x_i\}$:

- For each variable i the contribution s_i writes:

$$s_i = \log(P(x_i|C)) - \log(P(x_i|\neg C))$$

- For a new data-vector

$$\text{Evidence}(\mathbf{x}) = \log(P(C)) - \log(P(\neg C)) + \langle \mathbf{x}, \mathbf{s} \rangle \quad (1)$$

where $P(\cdot)$ is a probability, and $\langle \cdot, \cdot \rangle$ is a dot product.

Our parameter tuning heuristics for optimizing the generalization accuracy criterion, i.e., error percentage, (or F-score variant if necessary) is as follows:

- 0 – We start from the MIDOVA-expanded datatable, whose number of variables depends on our threshold parameters for the gain M_g and residue M_r indices, generally $M_g > 0$, $M_r = 0$.
- 1 – A first pass on the training set yields the ordered list of variables most contributing to the classification, sorted by decreasing s_i unsigned values.

- 2 – A 5-fold (or 6-fold) cross-validation on the training set yields the “optimal cut” I_{opt} for the number of relevant variables.
- 3 – A last pass on the whole training set, with parameter I_{opt} , yields the optimal value for the evidence threshold E_{opt} .
- 4 – The test set is then classified with formula (1) and parameters I_{opt} and E_{opt} .

IV. EXPERIMENTS

To our knowledge, public access test sets fitting to our requirements of qualitative datasets - binary or categorical – with two classes are uncommon. We present here a benchmarking of our classification method on three UCI repository dataset, All records containing unknown values have been removed. [9, 10, 11]: Tic-Tac-Toe, Wisconsin Breast Cancer, and Monks-2 [12] (known to be the most difficult of the three Monks problems).

As our aim in this paper consists in assessing the soundness of a novel data decomposition, and not in presenting a competitive learning algorithm, we have not tried to assess our MIDOVA application on the many other public access test sets with a k-class output variable ($k > 2$), or with numerical attributes to discretize. This way we avoid 1) further uncertainties in the comparisons due to the discretization steps, and 2) reprogramming other reference, or highly successful, methods, in our present proof-of-concept phase.

A. Tic-Tac-Toe

We have encoded the nine 3-category nominal variables (empty / cross / circle) into 27 binary variables, plus 2 binary variables for the class variable (“win/lose”). 638 instances are in the train set, 320 in the test set. The cross begins the game, and has to play when the given configuration instance appears.

The MIDOVA expansion yields 102 components, each including C or $\neg C$ (i.e., non C), with $M_g > 0$ and $M_r = 0$.

After reordering these variables, the Naïve Bayes parameter tuning, with a 6-fold cross-validation, keeps the $I_{opt} = 32$ most relevant ones, with threshold $E_{opt} = .8716$, resulting in the maximum, but yet attained, accuracy of 100% on the test set.

Note that a variant with 5-fold cross-validation results in 2 test errors (Accuracy=99.37%), and another one with $I_{opt} = 100$ results in 3 errors (Accuracy=99.06%).

Our method allows us to interpret the ordered list of the relevant itemsets: for example, the 4 top ones, with a prominent gain index of 168, encode the four diagonal patterns (O,X,O) and (X,O,X) associated with “lose” (the latter configuration is included in 68 instances, 42 “lose” and 26 “win”); the next 6 ones, with a gain of 144, encodes the trivial cases of three circles aligned in a row or a column, also associated with “lose”, and so on...

B. Wisconsin Breast cancer

This dataset consists in 683 patients (train set: 455; test set: 228) described along 9 ordinal scales. Eight of the scales have ten values, and one has nine ones.

For the sake of not losing the orderliness information, we have encoded each of the nine variables as follows: the i^{th} value is encoded by i “1” and $10-i$ “0” (for example, V1-3 results in {1 1 1 0 0 0 0 0 0}).

The MIDOVA expansion on these 89 binary variables yields 1283 components, each including C or $\neg C$, with $M_g > 0$ and $M_r = 0$.

After reordering these variables, the Naïve Bayes parameter tuning, with a 5-fold cross-validation, keeps the $I_{opt} = 130$ most relevant ones, with threshold $E_{opt} = .3189$, resulting in the maximum, not yet attained by explicit methods to the best of our knowledge, accuracy of 98.24% on the test set (4 errors). The recent reference [13] reports a 99.63% accuracy using a blind method (Artificial Metaplasticity Multilayer Perceptron)

Note that a variant with a standard binary coding scheme results in 5 errors (Accuracy=.9781).

Like any rule-based method, ours allows a medical expert to interpret the ordered list of the relevant itemsets, which top elements are:

Malignant←V2.5,V4.2	Malignant←V3.5,V6.4,V7.4
Malignant←V2.5,V7.4	Malignant←V3.5,V7.5
Malignant←V2.5,V4.3
Malignant←V6.4,V4.2	Benign←V2.5,V3.4
Malignant←V2.6,V4.2
Malignant←V6.8,V7.4	Benign←V1.4,V6.4,V7.4
Malignant←V2.5,V7.5

C. Monks-2

Monks2 is the harder of the three Monks problems: the solution cannot be described simply as a conjunction of disjunctions, it needs a method for pulling the concept of “exact number (n) of variables with value 1 amongst m ones” out of sample data.

We have encoded the six 3 nominal variables into 19 binary variables, plus 2 binary variables for the class variable (“two features/else”). 169 instances are in the train set, 432 in the test set.

The MIDOVA expansion yields 99 components, each including C or $\neg C$, with $M_g > 0$ and $M_r = 0$.

After reordering these variables, the Naïve Bayes parameter tuning, with a 5-fold cross-validation, keeps all of the $I_{opt} = 99$ of them, with threshold $E_{opt} = 1.6994$, resulting in the honorable accuracy of 71.5% on the test set: in the review [12], 9 symbolic learning techniques upon 24 result in a clearly better score. We are aware of only one SVM method [14] resulting in a better score (85.3%)..

Our method is clearly adapted to detecting classification rules expressed as conjunctions of disjunctions, not to more sophisticated hypotheses. But our experience is that this ability is enough for most of the real-life problems in the domain of supervised learning.

V. RELATED METHODS

Since the very beginning of this paper, we have continuously compared our method to Apriori: let us recall that our main objective here is to expose a novel representation of a 0/1 database, made of “salient” itemsets, close to the representation issued from Apriori, but with a very different definition of “salient”.

First, we will summarize the main similarities:

- Both are levelwise methods, starting from order-1 itemsets for building order-2, order-3, ... ones.

- Both aim at extracting the “local” information embedded in the interactions between two and more variables, resulting in a representation far from the global “datacloud” scheme of most of the data analysis methods.

- Both use anti-monotone properties: as regards to Apriori, the support of an itemset never exceeds the support of its subsets; concerning MIDOVA, the residue of an itemset never exceeds the residue of its subsets.

Then, the main differences are as follows:

- An implicit hypothesis needed by Apriori for giving rise to tractable computations is that each instance has a description of a “pick-any” type: it consists of a small number of items picked among a large number of potential ones, as it is the case for “market basket” data or language data – hence itemsets never include *absent* items in real-life applications. On the contrary MIDOVA is well-fit for any type of boolean data, whether pick-any or not, for it takes symmetrically into account both presence or absence of an item.

- The general principle of both methods are different: Apriori operates by counting the occurrences of combinations of the items, while MIDOVA condenses the 2^K facets of the huge K-way contingency table implicitly defined by any $N \times K$ boolean datatable into a list of its essential facets, where essential means “necessary and sufficient for rebuilding the datatable”.

- Both aim at minimizing the number of extracted itemsets, but Apriori uses frequency thresholds, while MIDOVA uses other criteria, preserving the cases where interesting itemsets may be unfrequent, if not absent (it is the case of the XOR function, and more generally of situations of exclusiveness – in a medical context, to characterize health may be as important as characterizing illness; see the *benign/malignant* example above).

- As a stopping criterion, Apriori uses support thresholds, while MIDOVA uses “residue”, a rigorous measure of the association potential of a considered itemset.

As regard to the illustrative part of our paper, which concerns the classification problem, we will review a few families of methods close to our classification scheme:

- The principles of the Association Rule-based Classifiers are as follows: 1) they start, as we do, from a Boolean matrix including the target variable, 2) they mine all the high-confidence association rules with a single variable in the right part – we use our novel MIDOVA process instead, 3) they filter the only rules implying the target variable, as we do, 4) they implement a rule-ordering strategy, generally based on support and confidence, instead of our Naïve Bayes-based expansion/selection process. The Large Bayes method of [6] is a salient reference in the 90’s. The references [15] and [4] report maximal accuracy rates of respectively 93.95% and 95.1% on WBC data, 92.6% and 98.2% on Tic-Tac-Toe, and no results on the Monks problems. Harmony [16] has taken over in the 2000’s. It uses an instance-centric rule generation framework where the ordering of local rule lists is based on confidence, entropy or

correlation criteria. At the end of the process and for each class, these lists are merged and sorted by the chosen criterion: when an unknown test instance is presented, the sums of the criterion for the k first relevant rules in each class are computed and compared, determining then the presumed class label. The recent reference [17] reports a 95.85% Harmony score for WBC data, and 97.98 for TTT. It presents a general scheme close to ours: the authors first create new features (based on frequency criteria, unlike ours) for expanding the data, then they use classic learning methods, among which Naïve Bayes, for the classification task. Their results with and without their “Feature Creation” (FC) expansion are reported in the recapitulative Table V.

Learning Classifier Systems (LCS) [5] are not as close to our method as it could seem at first glance: these *incremental* data-streaming algorithms use genetic optimization for the selection of best-fitted classification rules. This problem being harder than our batch-processing objective, no surprise that a 95,5% accuracy has been reported on WBC data [18].

TABLE V. REPORTED ACCURACIES FOR WISCONSIN BREAST CANCER, TIC TAC TOE AND MONKS2 DATASETS (^a: REPORTED IN [17])

<i>Method</i>	WBC	TTT	Monks2
Naïve Bayes	97.88 ^a	68.47 ^a	67.0
Naïve Bayes + FC	96.59 ^a	79.72 ^a	n.a.
Harmony	95.85 ^a	97.98 ^a	n.a.
(CBA (Classification Based on Associations)	93.95	92.6	n.a.
GARC (Gain based Association Rule Classification)	94.8	100.0	n.a.
LCS (Learning Classifier System)	95.5	n.a.	n.a.
Naïve Bayes + MIDOVA	98.24	100.0	71.5
Maximum reported performance with blind methods	99.58	100.0	85.3

VI. CONCLUSIONS AND PERSPECTIVES

We have presented in this text a novel representation scheme for qualitative data sets: a list of frequent and infrequent itemsets condensing all the noticeable, non-trivial information embedded in the interactions between Boolean variables. We have shown that this list is far less prone to the combinatorial explosion than the one resulting from the Apriori algorithm and that the maximum order of the interesting itemsets is limited to $\log_2 N + 1$, N being the number of instances in the database.

For proving the quality of this representation, we have decided to put it into practice in a supervised framework, in which a quantitative assessment is possible, specifically in the framework of the two-class discrimination problem. For this purpose, we have selected the subset of itemsets related to the class variable, resulting in an expanded datatable. We have chosen the Naïve Bayes classification method for providing the explicit discrimination criterion we wished and assess the quality of our data expansion.

The results on three open-access test datasets are satisfactory: we have proven on two test datasets (WBC and Tic-Tac-Toe) that, as a matter of accuracy performance, our derived classification method could compete with SVMs, while reaching the same human readability of the results as Learning Classifier Systems or Classification Association Rules. Honorable results were obtained on the Monks2 problem, which is an artificial test bench for general artificial intelligence.

Apart from developing non-supervised applications of our representation method, such as data-driven modeling, our middle-term prospects are many in the machine learning domain:

- Classify more than two classes and include splits on numerical variables, which could multiply our possible test benches, and outline more precisely the qualities and limits of our approach; we already know that on the Monks-2 dataset, our performance is good, but not excellent.
- Scale-up the implementation of our algorithm, for tackling real-life problems.
- Use another selection and classification method as Naïve Bayes, if necessary.
- Increase our theoretical understanding of the method, and bridge the gap with statistical learning approaches.

REFERENCES

- [1] Cadot M. and Lelu A. 2010. Optimized Representation for Classifying Qualitative Data. DBKDA 2010, pp. 241-246
- [2] Guermeur Y. 2002. Combining discriminant models with new multi-class SVMs. Pattern Analysis and Applications (PAA), Vol. 5, N. 2, pp. 168-179.
- [3] Naïm P., Wuillemin P. H., Leray P., Pourret O. and Becker A. 2007. Réseaux bayésiens, Collection Algorithmes, Eyrolles, Paris.
- [4] Chen, G., Liu, H., Yu, L., Wei, Q., and Zhang, X. 2006. A new approach to classification based on association rule mining. Decis. Support Syst. 42, 2 (Nov. 2006), pp. 674-689. (DOI= <http://dx.doi.org/10.1016/j.dss.2005.03.005>, 2012-06-01)
- [5] Bull L. (Editor), Bernada-Mansilla Ester (Editor), John Holmes (Editor), , 2008. Learning Classifier Systems In Data Mining, Springer
- [6] Meretakis D. and Wuthrich B. 1999. Classification as mining and use of labeled itemsets. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD-99).
- [7] Agrawal R. and Srikant. R. 1994. Fast algorithms for mining association rules in large databases, Research Report RJ 9839, IBM Almaden Research Center, San Jose, California.
- [8] Cadot, M. 2006. Extraire et valider les relations complexes en sciences humaines : statistiques, itemsets et règles d'association. Ph. D. thesis, Université de Franche-Comté (DOI= http://www.loria.fr/~cadot/cadot_these_2006.pdf , 2012-06-01)
- [9] UC Irvine Machine Learning Repository (<http://archive.ics.uci.edu/ml/>, 2012-06-01).
- [10] <http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29> , 2012-06-01.
- [11] <http://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame>, 2012-06-01.
- [12] Thrun S. and al. 1991. The MONK's Problems: A Performance Comparison of Different Learning Algorithms. S. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S.E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. (DOI= <http://robots.stanford.edu/papers/thrun.MONK.ps.gz>, 2012-06-01)
- [13] Marcano-Cedeño A., Quintanilla-Domínguez J., Andina D., 2010. Breast cancer classification applying artificial metaplasticity algorithm, Neurocomputing, Volume 74, Issue 8, pp. 1243-1250.
- [14] Rüping S. 2001. Incremental learning with support vector machines, in Proceedings of the 2001 IEEE International Conference of Data Mining.
- [15] Rajanish Dass. 2008. Classification Using Association Rules, IIMA Working Papers 2008-01-05, Indian Institute of Management Ahmedabad, Research and Publication Department. Downloadable in RePec: (DOI= <http://ideas.repec.org/p/iim/iimawp/wp02079.html>, 2012-06-26)
- [16] Wang J. and Karypis G. 2005. HARMONY: Efficiently Mining the Best Rules for Classification. SIAM International Conference on Data Mining, pp. 205-216
- [17] Gay D., Selmaoui-Folcher N., Boulicaut J.F. 2012. Application-independent feature construction based on almost-closedness properties, Knowledge and Information Systems, Volume 30, Issue 1, pp.87-111, Springer.
- [18] Wilson, S. W. 2002. Compact rulesets from XCSI, in Advances in learning classifier systems: Fourth international workshop, IW LCS 2001. (LNAI 2321), P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Springer-Verlag, pp. 196-208.

ANNEX

For the sake of comparability with other methods: K-itemsets extracted by MIDOVA out of the data in Table 1

k: #_items	k-itemset	Support	Gain	Residue	Frozen ?	Interesting ?	Interaction sign
1	{v01}	15	7.5	0	*		
1	{v02}	7	-0.5	7			
1	{v03}	7	-0.5	7			
1	{v04}	5	-2.5	5			
1	{v05}	6	-1.5	6			
1	{v06}	4	-3.5	4			
1	{v07}	6	-1.5	6			
1	{v08}	4	-3.5	4			
1	{v09}	5	-2.5	5			
1	{v10}	0	-7.5	0	*		
2	{v02; v03}	5	3	4		*	+
2	{v02; v04}	3	1	4		*	+
2	{v02; v05}	4	2	4		*	+
2	{v02; v06}	0	-4	0	*	*	-
2	{v02; v07}	1	-4	2		*	-
2	{v02; v08}	1	-2	2		*	-
2	{v02; v09}	1	-3	2		*	-
2	{v03; v04}	2	-1	4		*	-
2	{v03; v05}	3	0	6			
2	{v03; v06}	0	-4	0	*	*	-
2	{v03; v07}	0	-6	0	*	*	-
2	{v03; v08}	2	0	4			
2	{v03; v09}	2	-1	4		*	-
2	{v04; v05}	3	1	4		*	+
2	{v04; v06}	0	-4	0	*	*	-
2	{v04; v07}	1	-3	2		*	-
2	{v04; v08}	1	-2	2		*	-
2	{v04; v09}	2	-1	4		*	-
2	{v05; v06}	0	-4	0	*	*	-
2	{v05; v07}	1	-4	2		*	-
2	{v05; v08}	0	-4	0	*	*	-
2	{v05; v09}	0	-5	0	*	*	-
2	{v06; v07}	4	4	0	*	*	+
2	{v06; v08}	1	-2	2		*	-
2	{v06; v09}	2	0	4			
2	{v07; v08}	2	0	4			
2	{v07; v09}	3	1	4		*	+
2	{v08; v09}	3	2	2		*	+
3	{v02; v03; v04}	2	2	0	*	*	+
3	{v02; v03; v05}	2	-2	0	*	*	-
3	{v02; v03; v08}	1	0	0	*		
3	{v02; v03; v09}	1	0	0	*		
3	{v02; v04; v05}	2	0	4			

3	{v02; v04; v07}	0	0	0	*
3	{v02; v04; v08}	0	0	0	*
3	{v02; v04; v09}	1	0	0	*
3	{v02; v05; v07}	1	0	0	*
3	{v02; v07; v08}	0	0	0	*
3	{v02; v07; v09}	0	0	0	*
3	{v02; v08; v09}	0	0	0	*
3	{v03; v04; v05}	1	0	4	
3	{v03; v04; v08}	0	0	0	*
3	{v03; v04; v09}	1	0	4	
3	{v03; v08; v09}	1	0	0	*
3	{v04; v05; v07}	0	0	0	*
3	{v04; v07; v08}	1	0	0	*
3	{v04; v07; v09}	1	0	0	*
3	{v04; v08; v09}	1	0	0	*
3	{v06; v08; v09}	1	0	0	*
3	{v07; v08; v09}	2	0	0	*

Testing of an automatically generated compiler

Review of retargetable testing system

Ludek Dolihal

Department of Information systems
Faculty of information technology, Brno University of
Technology
Brno, Czech Republic
idolihal@fit.vutbr.cz

Tomáš Hruška, Karel Masařík

Department of Information systems
Faculty of information technology, Brno University of
Technology
Brno, Czech Republic
{hruska, masarik}@fit.vutbr.cz

Abstract— for testing automatically generated C compiler for embedded systems on simulator, it is necessary to have corresponding support in the simulator itself. Testing programs written in C very often use I/O operations. This functionality can not be achieved without support of the C library. Hence the simulator must provide the interface for calling the functions of the operation system it runs on. In this paper, we provide a method that enables running of programs, which use functions from the standard C library. After the implementation of this approach we are able to use the function provided by the C library with limitations given by the hardware. Moreover we add the overview of the testing system, which is used in our project. The system allows testing hardware and also software part of the project.

Keywords - Porting of a library, C library, compiler testing, simulation, hardware/software codesign, Codasip.

I. INTRODUCTION

This article is closely related to the paper [1] published at the ICCGI 2011. It will discuss the problematic of testing of the automatically generated compiler more closely, will focus on all major stages of compiler generation and on testing of the stages. As the main aim of the Lissom project [2] (commercialized under the registered mark Codasip - www.codasip.com) is hardware software codesign we have to test not just the software part but also the hardware part.

One goal of our research group is an automatic generation of C compilers for various architectures. Currently we are working on Microprocessor without Interlocked Pipeline Stages (MIPS). To minimize the number of errors in the automatically generated compilers, it is necessary to put the generated compilers under test. Because the whole process of the compiler generation is highly automatic and we do not have all the platforms, for which we develop, available for testing, we use simulators for compiler testing instead of the chips or development kits. In order to test the C compiler within any simulator, it is necessary to add the support for the C library functions into the simulator, which is used for the testing. The C programming language is still one of the most used languages for programming of embedded systems. Hence it is important to provide the reliable C compiler to the developers.

The support of the library is crucial in our project. We need to use tests written in C for the compiler testing and the tests commonly use I/O functions, functions for memory management etc. This paper presents the idea of fitting the simulator, where the testing is performed, with support of the C library and later on the implementation of this method.

The paper is organized in the following way. Second section provides the position of the testing in the Lissom project. After that we sketch the concept of retargetable testing system. Overview of the current stage of the testing is provided in section four. Then the short overview of related work is given, section six discusses the reasons for choosing the library. Sections seven and eight discuss theoretical and practical side of adding the library support into the simulator. Section nine describes the process of testing. Section ten presents the results obtained from commercial testsuite and finally section eleven concludes the paper.

II. RELATED WORK

As the core of the paper is dedicated to the testing of the compiler in the simulator we will focus mainly on related work in this area.

Simulators in general are one of the most popular solutions as far as embedded systems development is concerned. They are very often used for testing. We tried to pick up several examples that are connected to embedded systems development, and were published in a form of article. The Unisim project is not aimed at embedded systems but provides interesting idea.

Paper [6] presents a system that is very similar to the one that is developed within our project. It is called Upfast. The article describes system that generates different tools from a description file such as we do. The article mentions that C libraries were developed, but no closer information is given. It seems that in the simulator of the Unisim project the support for C language library have been right from the beginning. Unfortunately this is not our case. Porting of the library is critical for us, because without the support it is very difficult to test and evaluate the results of any tests.

Another interesting system including simulator is described in [7]. The project is called Rsim and is focused on simulation

of shared memory multiprocessors. The Rsim project works under Solaris. The Rsim simulator can not use standard system libraries. Unfortunately it is not explained why. Instead the Rsim provides commonly used libraries and functions. The Rsim simulator was tested for support of C library. All system calls in the Rsim are only emulated, no simulation is performed. In our system we will simulate the calls when necessary. The Rsim does not support dynamically linked libraries and our system also does not consider dynamic linking at the current state. Unfortunately in this article is not mentioned how the support for C library functions was added into the simulator.

Unisim project [8] was developed as an open simulation environment, which should deal with several crucial problems of today simulators. One of the problems is a lack of interoperability. This could be solved, according to the article, by a library of compatible modules and also by the ability to inter-operate with other simulators by wrapping them into modules. Though this may seem to be a little out of our concern the idea of the interface within the simulator that allows adding any library is quite interesting. In our case we will have the possibility to add or remove modules from the library in a simple way. But the idea from the Unisim project would make the import of any other library far easier than it is now.

The articles above are all related to simulations. The C programming language is not a new one and it is not possible to list all the articles that are in any way related to any library of C language. The different ways of compiler testing of any language are listed in [13]. The simulator is either created in a way that it already contains the library or it has at least some interface, which makes it easier to import the library in case it is wrapped in a module. Unfortunately our simulator does not contain such interface.

III. POSITION IN LISSOM PROJECT

In the Lissom project we focus mainly on hardware software codesign. In order to deliver the best possible services we want to provide the C compiler for a given platform as the C language is one of the main development languages for embedded systems. The C compiler is automatically generated from the description file. Besides the C compiler there are a lot of tools that are also generated from the description file. The tools include mainly:

- simulators,
- assembler,
- disassembler,
- profiler,
- hardware description.

The simulators can be generated either from a cycle accurate or an instruction accurate model. The profiler was thoroughly described in [3].

The description file is written in ISAC [4] language. The ISAC language is an architecture description language (ADL). It falls into the category of mixed ADL.

We would like to produce the whole integrated development environment for hardware software co-design. This IDE should provide all the necessary tools for developers when designing embedded systems from the scratch. The simulator is part of the IDE and C library support is part of the simulators (in the IDE can be more than one simulator).

The tool for generating compilers is called *backendgen* and is also embedded in the IDE. The quality of a compiler is crucial for the quality of software that is compiled by compiler. Hence it is very important to test the compiler that is generated by the *backendgen*. Via locating errors in the compiler itself we can afterwards identify and fix problems in the generation tools and in the whole process of development.

The *backendgen* closely cooperates with the semantic extractor. The semantic extractor as the title suggests, extracts the semantics of the instructions specified in the ISAC file and after that the *backendgen* creates backend of the compiler that recognizes given instructions. Both these phases of the compiler generation will be discussed later on.

The primary role of the C library is to enlarge the range of constructions that can be used during the process of testing. Testing of basic constructions such as if-statement, loops or function calls is important. On the other hand it is highly desirable to have a possibility of printing outputs or exiting program with different exit values and this can not be done without a C library support. The exit values are the basic notification of program evaluation and debugging dumps are also one of the core methods of debugging. Note that all the tests are designed for the given embedded system, and the tests are run on the simulator. The tests are aimed mainly on robustness of the system.

Secondary role of the library in the whole process of development is providing additional functions for writing programs. One of the most used functions is a group of functions used for allocating memory, string comparison and parsing, input/output methods etc.

As it is possible to generate several types of simulators in the Lissom project, it will be necessary to add the library support into all types of simulators. It should not include any substantial changes to the process of generation.

IV. CONCEPT OF THE RETARGETABLE TESTING SYSTEM

Forget about the technical details for a while and let us have a closer look at the concept of the testing system. We should define the goals we would like to achieve with our testing system. The Lissom project should have a robust system of testing that is built modularly. As the system should support hardware as well as software testing it should be composed of two main modules.

The very first question that should be answered is what parts of the project we need to test. The main aim and focus of this article is on the testing of the compiler backend. But there

are also other parts of the project that should be tested. The hardware realization of the chip, that was mentioned above is one of them. Also important is testing of tools that are not directly connected to the compilation toolchain, for example disassembler. This leads us to dividing the software module into two separate modules.

The testing system should be multiplatform and highly modular and also highly configurable. The addition of the new platform that should undergo the tests should be trivial. The microprocessors that we are going to test can vary in many ways. We need to support all these features of the microprocessors.

The task, for which the embedded system is going to be developed varies widely. On the other hand the tools that will be used for the development will stay more or less the same in all circumstances. This leads us also to the idea of the core system and many modules that should be optionally connected into the process of testing via interfaces.

As it was mentioned in the section 2 we can have either cycle or instruction accurate model. For the full testing we should have both of them. Full testing here means testing hardware as well as software part. Unfortunately it is not always possible. The testing system must reflect this and be able to adjust the testing to the actual conditions.

As far as the software testing is concerned we should take into account the different levels of compiler optimization as certain errors can be sensitive to this.

It is crucial to work with the most up to date tools so interface to any version system is a must. There should be also other interfaces, mainly the output ones. The system should be able to automatically inform a user about the result of the testing. There should be the email interface to send the result of testing to the person that performs it. We can also argue about interface to a bug tracking system such as Bugzilla or Trac. Though this interface would allow us to report the bugs automatically there is a risk of flood of false reports (the situation that one problem triggers others). Another issue is connected with the information that should be filled when the bug is created.

This problem could be solved by addition of a database between the testing system and the bug reporting tool. In the database we could keep records about the bugs that are currently reported and not yet fixed, hence we could avoid the redundancy of the bugs. Once the bug is fixed we could invalidate the database entry and if the same problem occurs again it could be reported again.

The notice about most up to date tools used for the testing leads to one module. The core module should be responsible for creation of all possible tools but not for testing of any kind. It should just verify that all the source code is valid and that tools can be created. Between the phase of creation of the tools and the testing of the tools is clearly defined interface. These two parts can be run separately.

Right from the beginning we should take into account that all our tools can be used under both UNIX and Windows

operation system. This is not a problem as far as the high programming languages are concerned (such as C or Java) for the programming of the devices. However the testing, which is the same in this case as running the testsuite should also be possible on both operation systems. And as the testsuite is created in Bash we must provide the basic support of the UNIX tools also under Windows. The solution here can be either MinGW or some other support such as Cygwin.

This brings us to the choice implementation language for the testing system. Unfortunately, the high level programming is not suitable for this kind of project. The testing involves an editing of various files, creating (make-ing the tools) control of return values and so on. Mainly for this reason, we chose the scripting in Bash as the best possibility. This brought us some difficulties as we will see later.

Our users will also use a different operation system and also different distributions of the UNIX systems. So from the beginning we must consider this. Not only different operation systems and also different releases must be taken into account as there might be different versions of the GCC compilers for example. The only way, we can sufficiently handle this is virtualization of the machines where the testing system will run.

At least some of the components of the testing system should be usable separately. It would be without all doubts useful to run just testing without the prior build of the tools. The tools can be built via the graphical interface for example. Dually, we can encounter a situation when the build of tools is sufficient and no testing should be performed. Arguably, the likelihood of the first case is higher. It is also given by the fact that there are several ways of building development tools.

Hence the module for the testing itself should stand alone and should have the clearly defined interface. For the thorough testing we should have as many tests as possible. Unfortunately, this goes against the principle of embedded systems. The microcontrollers often have very reduced instruction set, so the chips are not capable of executing the tests. Therefore, we need a system of the test selection that will ensure that just the clearly defined subset of tests will be compiled and executed for the given platform.

Hand in hand with the selection of the tests goes their evaluation. The selection of the tests should be centralized as much as possible. On the contrary, the evaluation of the tests can not be centralized thanks to the different testsuites we use in our project. They have different formats of the output and also exit codes differ in the meaning.

Together with the results and evaluation goes an issue connected with the reporting of the errors. Once we encounter an error and we want to report it we should know who is responsible for the error (or which tool generated the error). This could be determined via testing the tools separately. In case of testing all the tools together, we can rely just on error messages and on temporary files that could be created. By the temporary files we mean the files that are output of one tool and input of the very next tool.

V. OVERVIEW OF THE TESTING SYSTEM IN LISSOM PROJECT

At this point, I would like to give an overview of the testing system in the Lissom project. It should give the reader more precise information about the whole system and how the library fits into the whole. Our testing system is written in the Bash language. It consists of set of scripts. The testing system was originally developed for the UNIX systems. Should it also work under the Windows, it is necessary to support in the form of the MinGW. This approach brings on problems such as different paths on various systems or different settings of environment variables that have to be dealt with.

The testing system or testsuite as it is called in our project performs four basic tasks:

- testing of the tools for the development,
- testing of the backend of the C compiler,
- hardware testing,
- creation of the releases and packages of the models.

Now let us have a closer look at the parts of the project one by one.

A. Tools for the development

As far as the testing of the tools for the development is concerned it consists of several phases, which should be performed in given order.

As we always need to work with the most up to date tools the first thing that must be performed is the download of all necessary source code from repository.

The first phase is a build of all the tools. Even though the advanced IDE are used during the development very often happened that the source code can not be compiled.

Once the tools are created, the testing phase begins. We perform the testing of each tool and also testing of the toolchain to make sure that the cooperation is guaranteed.

Some of the tools such as assembler or simulator are platform dependent. So we have to keep in the repository the source codes for the testing for each platform. For the architecture independent tools this costs can be saved. The same problem occurs for the reference output. Certain tools can also have different levels of optimization and/or generation of information for profiling. Thanks to this fact the number of reference results grows rapidly. Currently, we are working on the new version of the testing system, and one of the tasks is to lower the number of reference outputs. Another weakness is that we do need the reference output. It is usually gained manually.

As mentioned earlier, we have different kinds of simulators. We perform testing on all kinds of simulators with all possible levels of generation of profiling information. The amount of generation of profiling information can be specified during the simulator generation. This is in contrast with testing

of the compiler backend, where we use just one simulator and generate minimal amount of profiling information.

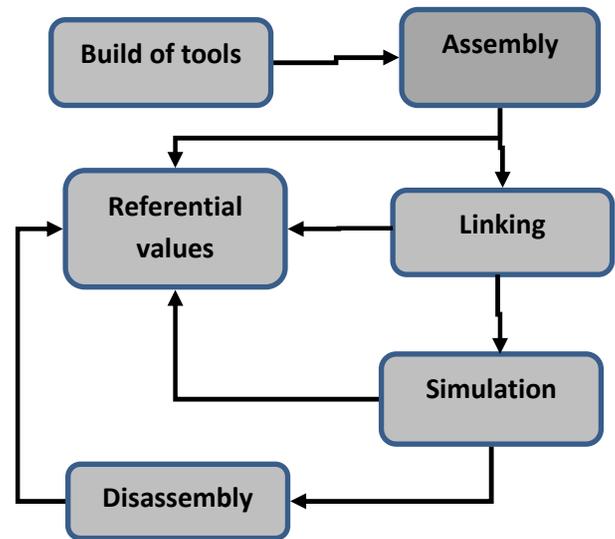


Figure 1. The scheme of testing of the development tools

We also perform tests that ensure the integrity of the whole system and a compatibility of the tools. In other words, we must ensure that if we add some new features into one of the tools the rest of them will be able to cope with these changes.

Typically, we bring some testing input written in an assembly language to the assembler and go through all the phases. In the end we should gain the executable file and be able to run it in the simulators with the correct return value. We also try to disassemble the executable. The code we receive should have the same functionality as the source one.

It may seem that both mentioned approaches are the same. However, the crucial difference is that while in the first case the tested component can go through the testing process without errors, there can be some issues connected with the file formats and interfaces between the tools. The first way of testing is on the other hand used for experiments with new features of particular components that are not supported by the whole toolchain yet. Figure one shows the process of testing of development tools. We have a simple program in C usually. This program goes through the whole toolchain. It is assembled, linked, simulated and in the end disassembled. After each stage we compare the result and referential value.

The hardware testing is also performed in this module. However we automatically perform just the tests of the syntactic correctness. No workbenches are executed.

B. C compiler backend

As far as testing of the compiler is concerned we first need to create the compiler and compiler driver. After that we can start testing. Here we will describe the process of the compiler generation and creation of compiler driver. The testing process itself will be thoroughly described later.

The LLVM project [10] is used by our research group as a base we build on. LLVM stands for low level virtual machine. It is a project focused on creation of modular compiler that provides aggressive optimization. In fact, the frontend and the middlend part of the compiler are used without massive changes. The part that is crucial from our point of view is the compiler backend.

The backend part is responsible for printing the assembler. This part is generated automatically by *backendgen*. As we use certain parts of the LLVM with no or small modifications we added the Lissom target into the LLVM project. This way we build programs that are later used for compilation of source code. Namely we create Clang this way. Clang is a frontend of the compiler provided by the LLVM project.

Given that we have built the LLVM project, we can start with the creation of compiler backend. This phase begins with the semantic extraction. As mentioned before the input of the whole process is file written in ISAC language. From the file that represents instruction accurate model of a microcontroller we extract semantics. After this phase, we get file that captures meaning of the instructions. More precise information about the semantic extraction phase can be found in the article [5].

The file with the extracted semantics is one of the inputs of the backend generator. The *backendgen* generates source files mainly in the C language that are later on compiled by ordinary C compiler (ie. gcc). As it is generated from the model it is clear that compiler backend is platform dependent. The semantic extractor and backend generator very closely cooperate. After the successful generation of the compiler backend we can create the compiler driver. The other tools that are required for the translation process are generated from the model before the backend is created. The brief overview of the backend generation can be found in here [5].

While the generation of all the tools is a must for the test compilation, it is not compulsory to build the compiler driver, but it simplifies the translation process considerably. The gcc compiler is in fact also compiler driver. We use compiler driver provided by the LLVM project. It is called *llvmc*. The tools that are used, parameters that are accepted by the tools and also the order of execution are described by the given syntax. The *llvmc* description has three parts. The first part is the description of the tools that are going to be used. In the second part, one must provide the languages (and its suffixes) that are the input and the output of each tool. Finally, we specify the relations between the tools. We can think of it as a graph. The tools became the nodes and we can think of the relations as of edges. The input and output languages are properties of the nodes.

We also use compiler-rt project of the LLVM. The compiler-rt project provides implementations of the low-level code generator support routines. This routines and calls are generated when a target does not have a short sequence of native instructions to implement a core IR operation. In fact, when the compiler does not know how to achieve certain behavior with the given instruction set it has a look at the compiler-rt library whether there is a call that could be used.

The main part of the library is composed around the floating point arithmetic. The functions have single float precision (which is denoted by the sf in the name of the function) and also double float precision (denoted by the df in the name of the function). As our processors do not usually have its own instructions for floating point arithmetic we very often use this library to provide the floating point emulation.

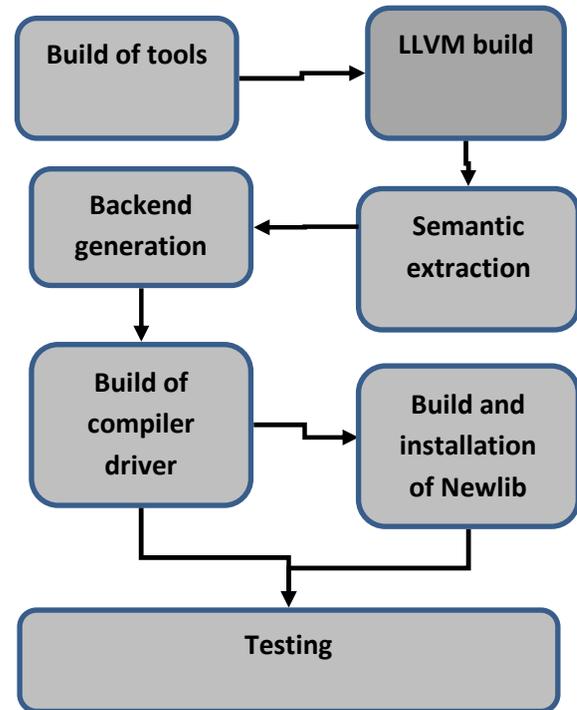


Figure 2. Scheme of testing of the compiler backend

The second figure shows in what order are the phases of backend testing executed. The libraries are not integral part of testing. It is possible to run the testing system without them.

The compiler-rt is for us just another library. We link it statically during the test compilation together with *newlib* for example. One of the issues is that this library is aimed at 32-bit systems. We would like to use it in simulators that simulate behavior of 16-bit processors. This has not been tested yet.

C. Packaging and releases

This module is a part of the testing system from the beginning. It was originally created for the building of the packages. As we currently support rpm distributions as well as deb distributions and also the Windows, the packaging system must reflect that.

The packaging system automatically creates the packages for the currently supported platforms. The package includes all the tools that are needed for the development on a given platform. Currently we support Ubuntu and Debian releases, Fedora, CentOS, OpenSUSE and Windows 7. For the majority of the UNIX distributions, we maintain the current release and previous one. All this systems run as virtual servers. The

created packages are automatically uploaded at our web pages, where can be downloaded by our co-developers and users.

Later also the packaging of the models and the model documentation were added. These are also available from the web pages. We have also started with nightly builds to ensure, that the committed changes do not affect the build in a negative way. Yet another advantage of nightly builds is that if any package is needed with the changes that were made within the last 24 hours the package was already created overnight and we do not have to wait for it to build. We can be uploaded it at the server where it is available to the customers.

D. Stability of the system

For a long time, we had problems with the stability of the whole testing system. As it is composed purely of the Bash scripts, we very often faced the problem, that one part of the testing system broke down according to an error but the build went on and the error was lost deep in the logs. This was typical situation during the night build, when the system is unobserved. In the morning we realized, that the packages were not created and started to look for the reasons. As our system creates a lot of logging information it was not always easy to identify the reason.

To solve this problem we decided to create simple wrapper and wrap all the commands except the calls of our procedures and functions. The wrapper performs the command that is given to it as a parameter and controls various variables. Clearly one of the most important is the return value of the command. If the return value is out of range we simply call system exit and the whole process stops with the clearly specified error message.

Even more important is the fact that we know the exact place where the error occurred. Unfortunately, this is not true in case we apply parallel build. But the wrapper is applied on the Bash commands so we at least know the command where the error occurred hence we can narrow the area and focus on the command more precisely. After the application of the wrapper the stability of the whole system improved.

VI. CHOOSING THE LIBRARY

As we are focused mainly on embedded systems and we design the whole process of compiler development for them we dedicated quite a lot of time to choosing the correct library. It was clear right from the beginning that glibc is needlessly large and therefore not suitable for use in embedded systems. We need library that satisfies following criteria:

- minimalism,
- support for porting on different architectures,
- well-documented,
- new release at least once a year,
- compatibility with glibc,
- modularity.

All these conditions were satisfied by few libraries. Amongst those we chose Newlib [9]. This library is largely minimalistic. It does not contain certain modules, because, according to the authors, it would be against the minimalism. In certain areas it sacrifices better performance in favor of minimalism. For example functions for I/O could be optimized for different platforms, but there is just one version for all platforms written in portable C that is optimized for space.

As far as the new releases are concerned, it can be said that the library is alive. New version is released at least once a year. This is very important because we need to keep pace with the up to date versions of glibc. There are other minimalistic libraries compatible with glibc, but quite a lot of them are not maintained sufficiently.

Another reason for choosing the newlib is the documentation that is provided with the library. Whole process of porting the library to different platform is well-documented and thanks to the wide use of the library it is not difficult to find help.

The most important reason for choosing the newlib is the fact, that it has already been ported to several platforms. One document is dedicated to the process of porting and even though we do not port the library to new architecture it can provide us with very useful information. During the process of porting we will perform steps that are similar to porting the library to any new architecture.

Unfortunately this library is dependent on kernel header files. But during the porting we will get rid of these dependencies. We will need to use this library under UNIX systems as well as under Windows.

VII. THEORY OF PORTING

The main reason for porting the library into simulator is the fact that we need to add the support for C functions into the simulator itself. To be precise, we want to use the libc functions such as printf, malloc, free etc. in the programs that will be used for testing of the compiler. And because we do not possess the development kits for all the platforms we use simulators instead.

If one does not grant libc library support in the simulated environment, the number of constructions we can use and test is very limited.

Consider the following simple example written in C:

```
int main(int argc, char **argv)
{
    if(strcmp("alpha","beta")==0)
    { return 1;}
    else
    { return 0;}
}
```

Even this simple program can not be executed, because it uses function `strcmp` that is part of the C library. This program can not be compiled unless the inclusion of `string.h` and possibly some other header files is included.

On the contrary the main aim of testing is to cover as wide area as possible and also try as many different combinations of functions as we can. However, this goes against the idea of embedded solutions. And because we focus especially on embedded systems, we do not even try to cover all the functions provided by `glibc` or in our case `newlib`. In fact we will use and hence test only functions that can run under the simulated environment and are useful for the programs that will be executed on the given platform. Moreover embedded systems are not designed for use of vast number of constructions that programming languages offer. Typically there is just one task, usually quite complicated, that is launched repeatedly. As we will see the functions that we will use form just small part of `newlib`. The functions that are not important to us can be easily removed via configuration interface or it is possible to remove them manually. Following categories are examples of unimportant functions:

- threads, we assume that in simple programs for embedded systems one will not use threads,
- locales, all the locales were removed from the library,
- math functions for computing `sin`, `cos` etc.
- `inet` module, even though networking plays important part in modern embedded systems whole module was removed,
- files and operations with files, our application do not need interface for working with files.

Now we come to the important parts of the library. Simply spoken all that really has to remain from the library are the `sysdeps`, this is the core of the whole system (how to allocate more memory etc.), then important modules such as `stdio` (for outputs, inputs) and other modules we wish to preserve. In our case we wished to preserve following parts of the `newlib` library:

- `stdio`, this was the main reason for porting the library, to get in human readable form output from the simulator,
- module for working with strings and memory, in our applications we would like to use functions such as `memcpy`, `strcpy`, `strcat` etc.,
- memory functions, for example `malloc`, `free`, `realloc`,
- `abort`, `exit`,
- support for `wchar`, but without support of different encodings.

Some parts of the library could not be removed because of the dependencies. According to our estimations nearly 40 percent of the library was disabled or removed measured by the size of the library.

There are several ways of building the library and also different methods of using it. There is a possibility of building a position independent code. Even though this is an interesting solution we decided against it. Instead of PIC (position independent code) we are going to compile the library into single object and then link it to the program statically. The position of library in the whole process of testing is shown in the figure 3. The library is linked to the program and after that the program is loaded into the simulator.

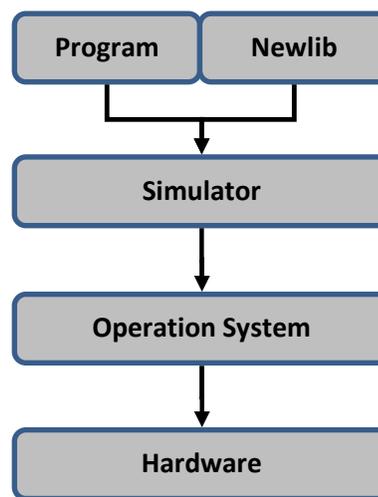


Figure 3. Scheme of calling `printf` function

Now return to the functions that remain in the library. They can be divided into two groups. First group consists of functions that are completely serviced within the simulated environment. For example function `strcmp` falls into this category. This function and its declaration remains unchanged within the simulator if it is written in portable C. These functions are not tied with kernel header files so there is no need to change them.

The second group of functions consists of functions that are translated to the call of system function. Function `printf` can be used as an example of this group of functions. The call of `printf` function can be divided into three phrases that are illustrated at the following picture.

In the beginning the call of `printf` function is translated on the call of system function, with the highest probability it is going to be the call of function `Write`. `Write`, being the POSIX function, is offered by the operation system. But as we want to use the simulator on UNIX platform as well as on Windows systems we have to remove these dependencies. To do so we will use the special instruction principle.

A. Use of ported library of UNIX and Windows systems

Before we get to the principle of special instruction method we should explain why we need to use this method. The main reason why we should remove the dependencies on the kernel header files is the fact, that we must be able to use the library

under UNIX systems and also under Windows like operation systems.

As long as we use the library under UNIX systems everything should be all right. Though even on UNIX systems there might be differences amongst the different versions of the header files. But once we use the Windows based system we can not use header file functions any more. It would almost certainly result in a crash of the system.

In our project we currently support several UNIX distributions as well as Windows. Use of other operating systems is not considered.

B. Special instruction principle

The special instruction principle means, that we will use instruction with the opcode that is not used within the instruction set for the special purpose. So far all architectures that were modeled within the Lissom project had several free opcodes. It is typical that the instruction sets do not use all operation codes that are provided. But in case of no free opcode this method can not be used. The special instruction principle will be used for ousting the dependencies on kernel header files.

Functions provided by operation system are called by the syscall mechanism. The system calls can be quite easily detected. Each library should have defined the syscall mechanism in special source file. This syscall mechanism differs, as they usually are platform dependent. So i386 architecture will have different syscall mechanism than arm.

Syscalls together with other code that is platform dependent are kept in a specific folder. When the library is compiled, the platform dependent code is kept in a special archive and is separated from the platform independent code. Figure 4 shows this situation. We must link two different archives to the program we wish to execute. The C library and the archive containing syscalls and other platform dependent code such as runtime etc.

We wish to preserve the mechanism. The syscalls will remain in the library, but with different meaning. The file containing syscall will be changed in the following way: in the beginning the parameters of the syscall will be placed at the given addresses in the memory and we will also define where the syscall return value will be placed. Afterwards the call of the chosen instruction will be performed. It is also possible to put the parameters into registers, but some platforms have limited number of registers, hence this method could cause problems.

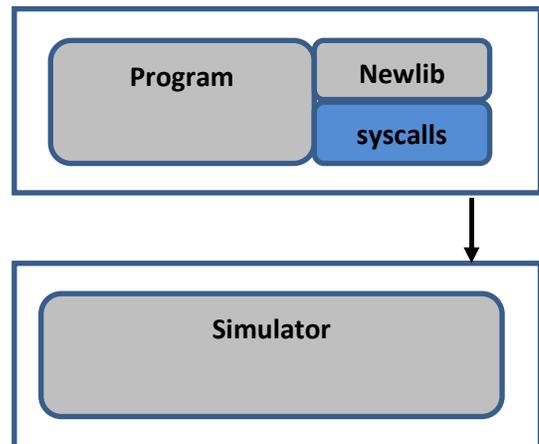


Figure 4. Scheme of calling the simulator via newlib layer

The syscall mechanism is in fact a wrapper of the system call. The call will be passed to the simulator that will do the call and return the result.

C. Simulators

As was mentioned before, all the simulators are generated automatically. In the beginning all the source files are generated by specialized tools. When the generation phase is finished the simulator is build by a Makefile. It will be necessary to add into this process following information:

- information about which instruction (opcode) calls the system function,
- the simulator will have to know the convention for storing parameters,
- the simulator will have to recognize which system function is going to be called,
- the simulator will have to perform the call of the correct system function.

First three points will be solved within the model of an instruction set. The instruction with the opcode that is not used will be declared. The instruction behavior will be defined in the following way: according to the parameters it will call the given system function. The simulator will have to recognize the system it runs under and call the correct function. For example on UNIX system it will be function write and in Windows WriteFile. This should be solved by the libc library of the given platform. The following figure demonstrates the call of special instruction.

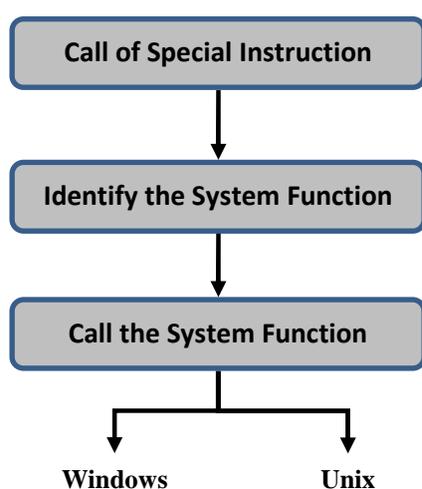


Figure 5. Calling sequence of specialized instruction

When the special instruction is called, we need to identify, which system function we need to execute. This information must be passed out of the simulator.

The parameters that were placed at the given position at the simulated memory can remain unchanged. They will be later passed to the specific system call.

One important issue is connected with the simulated memory. As we would like to correctly simulate the operations with memory such as malloc, realloc etc. we need to tell the simulator how many memory it can simulate. This will be done by the special file that will be passed to the linker. This file will contain symbols that will declare how much memory can be used.

We also considered completely different attitude to this problem. Instead of monitoring calls of system function we could monitor memory accesses. But it would slow down whole process of simulation.

VIII. PROCESS OF PORTING

Before the whole process of porting begins we need to download the newlib. There are two possibilities. It is possible to download only the library or there is a whole toolchain for development of embedded system for given architecture, so called buildroot.

The main advantage of downloading the whole buildroot is that once it is built you get whole set of development tools including various compilers, linkers, debuggers, strip programs etc. You also get the build of newlib. These tools are quite useful in the beginning when you remove unwanted modules from the library, because they can be used for rebuilding the library.

One of the problems we faced is that we need to have the compiler for the architecture we are developing for. In other words if we want to create a library for testing C compiler on a given platform we need a compiler for the same platform that is already created. The compiler will be used for building the newlib. Moreover the compiler must have exactly the same

instruction set. In the future we would like to use the generated compiler for building the library. This requires high quality of *backendgen* and generated backend.

Because we are going to use the library in the simulator and the simulator can handle only instructions of the specified instruction set, then the library must be translated to the instruction set that is recognized by the simulator. For building the simulator we can use common gcc for Windows or UNIX, because it runs under common system.

This may be the first big problem in the whole process of porting. It is not hard to find a compiler for given platform. Nowadays there are specialized compilers for nearly all architectures used in embedded systems. The buildroot for newlib contains more than dozen of different architectures such as MIPS, arm, mipsel, sparc etc. There are even different versions of the mircoarchitectures in case of MIPS for example.

Problem is that thanks to the aim of the whole Lissom project, there we usually use specialized instruction sets or we use some generic instruction set and add certain specialized instructions. After this customization it is usually impossible to use generic compiler for building the library.

We could use for building the library the compiler that we want to test but currently it is not stable enough for building large programs. The best solution of this problem is usually building a specialized toolchain including GNU binutils and GNU compiler collection. As was mentioned once the generated backend is stable enough it will be used for building the library.

Several issues we faced during the process were closely related to the buildsystem of the library. The library contains a system of makefiles. This system is hierarchical and usually the makefiles from the upper levels are included. So if for example we would like to compile any test examples that are included in the newlib we switch to the given directory and call make. This will call all the makefiles from the above directory. This is very effective, because only the makefile in the root directory contains variables defining which compiler, assembler, linker will be used. On the other hand it is very difficult to modify this system in case we want to build the different parts of the library using different tools.

Currently we are using for the development the set of our tools containing archiver, linker, assembler and compiler. The currently used compiler is called mips-elf-gcc. It is not generated automatically but was created especially for this purpose as our generated compiler is not yet stable enough. Linker and archiver are not generated automatically but were developed in Lissom project.

Our tools are not compatible with the tools that were originally used for building the library. Our tools do not support so wide variety of parameters so some of them had to be erased from the configuration files and some were just changed to suit our needs.

Currently we use set of scripts, which preprocess the flags. In the scripts we erase the flags we do not need and do necessary substitutions.

The buildsystem of the library starts by parsing the configuration file and accord to the content of the file are set different macros and variables. When doing manual changes to the buildsystem we have basically two possibilities:

- change the configuration file or,
- do the changes later in the Makefiles.

The first possibility is cleaner but the Makefiles often check if the option is present in the configuration file and ends with error in case the option is missing. Hence it is more convenient to do the necessary changes in the Makefiles. Thanks to the hierarchical structure it is in most cases sufficient to do the change in just one place.

We also use different formats of the output files. Output of our assembler is an object file .obj that is not compatible with .o that is the usual output of gcc compiler. Currently we use mips-elf-gcc just for compilation from C to assembler. After this phase we use automatically generated assembler to compile the files from assembly language to object files that are later used by the archiver.

In the theoretical part we mentioned the need to link special file containing information how much memory can be used. The file will contain symbols defining the beginning and the end of memory space that can be used. It will have the following syntax:

```
#file defining memory boundaries
define start 256
define stop 768
```

Given that the numbers are in kB the simulator can simulate up to 512 kB of memory. Character # denotes comment.

As far as the convention for storing parameters is concerned, we have chosen following approach: first parameter says, which system function is going to be called. In the newlib it is a list of system functions for UNIX systems. The rest of the parameters (2-7) are passed to the function call. The parameters remain unchanged. They are passed to the system function in the exactly same state, in which were saved in the memory before calling the special instruction. The special instruction itself has no parameters. When the instruction is called, all the parameters have to be stored in the memory at given addresses.

A Automation of the porting process

As for the first time all the steps were performed manually. In the future we would like to automatize this process as much as possible. Without doubts we could remove the needless parts of the library automatically. The needless parts would be identified by the configuration file and also the special instruction principle could be highly automatic. If we have spare instruction we will choose it and compose it into the

simulator. Unfortunately there are steps that need to be performed manually. For example we need to provide the runtime for the simulators and the corresponding sections needs to be specified in the ISAC file.

Runtime is also one of the files that are written by hand in assembler. There are also other files written in assembly language and hence are platform dependent. In case of MIPS platform there were 8 files that contained assembly language. For example syscalls or memcpy functions are ale implemented in assembler. In order to minimize number of files written by hand we decided to provide as much files written in portable C as possible. We managed to replace all but two files by C implementations. All that have to be provided is the runtime and syscall mechanism.

IX. PROCESS OF TESTING

Now when we have thoroughly gone through the library porting, we can have a look at the test selection issues.

A. Test selection phase

As we have a large amount of tests from the different sources (gcc-testsuite, llvm-testsuite, etc.), we need a universal approach that will define, which tests are suitable for the compilation and execution on a given platform.

We have created a system of files that restricts the number of the tests that can be compiled on a given platform according to the libraries that are available. The libraries are just one of the test selection criteria; also other characteristics are taken into account for example the size of the registers or the size of stack.

The naming convention for these files is very simple. The file bears same name as the test does but have suffix .x instead of .c. The system is hierarchical. We can have the hierarchy because we support a nesting of the directories and we keep .x files not just for the tests, but also for the directories. In case of the directory the .x file has the same name as the directory with the .x suffix.

These files have minimal functionality. We try to keep their size as small as possible. Their typical functionality is that according to some state of the flags the test is excluded from testing, because implicitly all the directories and all the tests are selected for the testing. So, if we want to exclude the tests or whole directories from testing we have to indicate this.

As the size of the files is kept minimal the functionality and flag settings must be done elsewhere. This is performed centrally in the main testing module. The functions that check the current state of the flags and control what libraries are accessible for linking to the given platform are declared here. The centralization in this case has purely practical base. The typical usage of the .x files is that we disable testing of the whole directories according to the libraries that are accessible. The .x files can also bear other functionality. We can for example set different variables. We can specify flags that should be added to the compilation or add some files to the linker as in the following example.

```
if [ "$C_LIB" == "0" ]; then
    FILE_DEPS+=crt0.o
fi
```

On the level of files we most often use the .x files for the filtering the test that depend on the compiler-rt library for a given platform. As usually only few tests of any directory depend on the compiler-rt and the dependence does not have to be same for all platforms, the best solution is to condition the test execution by the platform and the compiler-rt presence. This is demonstrated in the following example.

```
is_arch "mips_basic" $1
if [ "$?" == "0" ]; then
    if [ "$RUNTIME_LIB" == "0" ]; then
        RUN_TEST=0
    fi
fi
```

The presence of certain libraries can be also criteria for testing because some tests have library dependencies. The biggest advantage of this approach and also the main reason for introduction of this system is its universality. We employ the tests from the LLVM testsuite, the gcc testsuite, the Mibench [11] set of tests and we also have tests that were created within our project. The system of the .x files can be used for all these sources as long as we use just the tests without the testing infrastructure that is provided in several cases.

The only set of tests that we use together with the infrastructure that is provided together with the tests is the Perennial testsuite [12].

B. Test compilation and execution

The compilation of the tests is performed in the central module. As we have the system of the .x files we enter only those directories that are suitable for the testing on the given platform. So before entering the directory with tests we check the .x file for a given source and consult the restrictions that are defined by the .x file and set all the variables denoted by the file.

If the directory is feasible for testing we cycle through the tests in an order denoted by the test list. The .x file is always checked first, and if nothing blocks the test it is compiled. The presence of the .x files is not compulsory. As mentioned earlier the default setting is to go through all directories and execute all tests. But if the file is present it will be checked.

If there are any problems during the test compilation they are logged. We keep the list of the tests that were not compiled successfully together with the output of the compiler. The logs are kept for every platform that is tested to avoid an overwriting. It is also possible to create unique log not just for each platform but for every run of the testing system. These logs could be in the future stored in the database to keep precise testing history. The tests are compiled and executed several times with different levels of compiler optimization. Currently we support levels from 0 up to 4.

C. Logging information and test evaluation

The test evaluation is kept decentralized. As we deploy tests from different sources we need to keep the test evaluation together with the tests. For some tests we evaluate on the basis of exit code, but there are the tests that produce for example text output and we have to compare the output with the referential values (this is where the library comes to use).

The decentralization in this case means that we keep for every directory a shell script that takes care of test execution and evaluation.

As in case of test compilation we keep detailed logging information. We keep the output of the simulator and after the test evaluation we list it into the list of passed tests or failed tests according to the result of evaluation. The logs are created for every tested platform and can bear time reference.

X. RESULTS OF PERENNIAL TESTSUITE

For having a comparison with commercial compilers we tested our automatically generated compiler with commercial Perennial testsuite. The results described here were gained from the generated MIPS compiler.

The testing was performed on our complete toolchain. The tests were compiled by our generated compiler and afterwards executed the tests on our simulator that was also automatically generated.

We have only part of the Perennial testsuite. We used only the tests that examine the core of the compiler. We excluded some of the tests that can not be compiled due to the header files dependencies we do not support. The tests in the testsuite are divided into groups according to the chapter of the standard that is tested. We use tests for the clauses 5 and 6. We have mainly tests for the standard C90 and several tests for C99 standard. Currently we have no tests for C11 standard.

The final number of tests that we execute is 796. From 796 tests are 794 tests compiled and executed correctly. Only two tests fail either during the compilation or return incorrect value. The results are summarized in the following table.

Table 1: Results of the Perennial testsuite

Compiler	All tests	Pass tests	Fail tests	Not compiled	Not executed
Lissom	796	794	2	2	0
Gcc	796	796	0	0	0

As the table shows, just 2 tests do not succeed. After closer look we realised that this two tests use trigraphs, that are not supported in the llvm frontend. This tests can not be compiled by the current version of the llvm. The tests were compiled with O2 optimization.

The table also provides comparison with gcc compiler for i386 platform. The gcc compiler in version 4.6.3. compiles all the tests and the programs are executed correctly. We were also interested in how much time does the program spend by syscall execution. We compiled for our platform a program that accomplished MPEG decoding. The input and output

streams of the program were redirected into the files. The profiling of the MPEG decoder showed, that execution of syscalls took less than 2% of time.

XI. CONCLUSION

In this paper, we gave the overview of the testing system in our project and sketched the idea of adding the support for the C library into the simulator. The motivation is quite clear: to be able to use the library functions in the tests that are run on the simulator of the given microcontroller. The special instruction principle was proposed, which enables us to forward the call of system function. It also allows us to identify, which system function is called. This principle is quite universal and can be used for the majority of platforms. After implementation of this method, we are able to run all the functions that are commonly used such as I/O functions, memory management and string functions, etc. Moreover, we can adjust the library according to our needs. Thanks to the modularity we can enable or disable any module. This may turn to be an advantage, because the complete library occupies tens of megabytes and the compilation and linking such a library can be time consuming.

We also tested our generated compilers with the commercial Perennial testsuite. We had only chosen a subset of tests that should validate the core of the compiler. The compiler was tested against the C90 and C99 standard with good results when we take into account the fact, that the compiler is generated automatically. The fact that we can easily compose new testing systems into our own together with the results we gained is encouraging.

ACKNOWLEDGEMENTS

This research was supported by doctoral grant GA CR 102/09/H042, by the grants of MPO Czech Republic FR-TII/038, by the grant FIT-S-11-2 and by the research plans MSMT no. MSM0021630528. This work was also supported by the IT4Innovations Centre of Excellence Project CZ.1.05/1.1.00/02.0070 and by the Artemis EU Project SMECY.

REFERENCES

- [1] L. Dolihal and T. Hruska, "Porting of C library, Testing of generated compiler", In proceedings of ICCGI 2011, Jun. 2011, pp.125-130,
- [2] Lissom Project. <http://www.fit.vutbr.cz/research/groups/lissom> [online, accessed: 18.6.2012]
- [3] Z. Prikryl, K. Masařík, T. Hruška, and A. Husár, "Generated cycle-accurate profiler for C language", 13th EUROMICRO Conference on Digital System Design, DSD'2010, Lille, France, pp. 263–268.
- [4] K. Masarik, T. Hruska, and D. Kolar, "Language and development environment for microprocessor design of embedded systems", In proceedings of IFAC workshop of programmable devices and embedded systems PDeS 2006, pp. 120-125, Faculty of electrical engineering and communication BUT, 2006
- [5] A. Husar, M. Trmac, J. Hranac, T. Hruska, and K. Masarik, "Automatic C Compiler Generation from Architecture Description Language ISAC", Sixth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'10) -- Selected Papers, pp. 47-53.
- [6] S. Onder and R. Gupta, "Automatic generation of microarchitecture simulators," Computer Languages, 1998. Proceedings. 1998 International Conference on , vol., no., pp.80-89, 14-16 May 1998
- [7] C.J. Hughes, V.S. Pai, P. Ranganathan, and S.V. Adve, "Rsim: simulating shared-memory multiprocessors with ILP processors ," Computer , vol.35, no.2, pp.40-49, Feb 2002,
- [8] D. August, J. Chang, S. Girbal, D. Gracia-Perez, G. Mouchard, D. Penry, O. Temam, and N. Vachharajani, "UNISIM: An Open Simulation Environment and Library for Complex Architecture Design and Collaborative Development," Computer Architecture Letters , vol.6, no.2, pp.45-48, Feb. 2007
- [9] newlib. <http://sourceware.org/newlib/> [online, accessed: 18.6.2012]
- [10] C. Lattner and S.V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation", Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04), Palo Alto, California, Mar. 2004
- [11] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, "MiBench: A free, commercially representative embedded benchmark suite", Workload Characterization, Dec. 2001, pp.3-14, doi:10.1109/WWC.2001.990739
- [12] Perennial testsuite, <http://www.peren.com/> [online, accessed: 18.6.2012]
- [13] A.S. Kossatchev and M.A. Posypkin, "Survey of compiler testing methods", Programming and Computer Software, Jan. 2005, pp.10-19, doi: 10.1007/s11086-005-0008-6

Compiler-based Differentiation of Higher-Order Numerical Simulation Codes using Interprocedural Checkpointing

Michel Schanen, Michael Förster, Boris Gendler, Uwe Naumann
 LuFG Informatik 12: Software and Tools for Computational Engineering
 RWTH Aachen University
 Aachen, Germany
 {schanen, foerster, bgendler, naumann}@stce.rwth-aachen.de

Abstract—Based on algorithmic differentiation, we present a derivative code compiler capable of transforming implementations of multivariate vector functions into a program for computing derivatives. Its unique reapplication feature allows the generation of code of an arbitrary order of differentiation, where resulting values are still accurate up to machine precision compared to the common numerical approximation by finite differences. The high memory load resulting from the adjoint model of Algorithmic Differentiation is circumvented using semi-automatic interprocedural checkpointing enabled by the joint reversal scheme implemented in our compiler. The entire process is illustrated by a one dimensional implementation of Burgers' equation in a generic optimization setting using for example Newton's method. In this implementation, finite differences are replaced by the computation of adjoints, thus saving an order of magnitude in terms of computational complexity.

Keywords—Algorithmic Differentiation; Source Transformation; Optimization; Numerical Simulation; Checkpointing

I. INTRODUCTION

A typical problem in fluid dynamics is given by the continuous Burgers equation [2]

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \quad , \quad (1)$$

describing shock waves moving through gases. u denotes the velocity field of the fluid with viscosity ν . Similar governing equations represent the core of many numerical simulations. Such simulations are often subject to various optimization techniques involving derivatives. Thus, Burgers' equation will serve as a case study for a compiler-based approach to the accumulation of the required derivatives.

Suppose we solve the differential equation in (1) by discretization using finite differences on an equidistant one-dimensional grid with n_x points. For given initial conditions $u_{i,0}$ with $0 < i \leq n_x$ we simulate a physical process by integrating over n_t time steps according to the leapfrog/DuFort-Frankel scheme presented in [3]. At time step j we compute

$u_{i,j+1}$ for time step $j + 1$ according to

$$u_{i,j+1} = u_{i,j-1} - \frac{\Delta t}{\Delta x} (u_{i,j} (u_{i+1,j} - u_{i-1,j})) + \frac{2\Delta t}{\Delta x^2} (u_{i+1,j} - (u_{i,j+1} + u_{i,j-1}) + u_{i-1,j}), \quad (2)$$

where Δt is the time interval and Δx is the distance between two grid points. In general, if the initial conditions $u_{i,0}$ cannot be accurately measured, they are essentially replaced by approximated values. To improve their accuracy additional observed values $u^{ob} \in \mathbb{R}^{n_x \times n_t}$ are taken into account. The discrepancy between observed values $u_{i,j}^{ob}$ and simulated values $u_{i,j}$ are evaluated by the cost function

$$y = \frac{1}{2} \sum_{i=1}^{n_x} \sum_{j=1}^{n_t} (u_{i,j} - u_{i,j}^{ob})^2 \quad , \quad (3)$$

which allows us to obtain improved estimations for the initial conditions by applying, for example, Newton's method [4] to solve the data assimilation problem with Burgers' equation as constraints [5]. The single Newton steps are repeated until the residual cost y undercuts a certain threshold.

In Section II, we introduce Algorithmic Differentiation (AD) as implemented by our derivative code compiler `dcc` covering both the tangent-linear as well as the adjoint model. Section III provides a user's perspective on the application of `dcc`. Higher-order differentiation models are discussed in Section IV. Finally, the results of our case study are discussed in Section VII.

II. ALGORITHMIC DIFFERENTIATION

The minimization of the residual is implemented by resorting to Newton's second-order method for minimization. In general, Newton's method may be applied to arbitrary differentiable multivariate vector functions $\mathbf{y} = F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$. This algorithm heavily depends on the accurate and fast computation of Jacobian and Hessian values, since one iterative step $\mathbf{x}_i \rightarrow \mathbf{x}_{i+1}$ is computed by

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \nabla^2 F(\mathbf{x}_i)^{-1} \cdot \nabla F(\mathbf{x}_i) \quad . \quad (4)$$

The easiest method of approximating partial derivatives $\nabla_{x_i} F$ uses the finite difference quotient

$$\nabla_{x_i} F(\mathbf{x}) \approx \frac{F(\mathbf{x} + h \cdot \mathbf{e}_i) - F(\mathbf{x})}{h}, \quad (5)$$

for the Cartesian basis vector $\mathbf{e}_i \in \mathbb{R}^n$ and with $\mathbf{x} \in \mathbb{R}^n$, $h \rightarrow 0$. In order to accumulate the Jacobian of a multivariate function the method is rerun n times to perturb each component of the input vector \mathbf{x} . The main advantage of this method resides in its straightforward implementation; no additional changes to the code of the function F are necessary. However, the derivatives accumulated through finite differences are only approximations. This represents a major drawback for codes that simulate highly nonlinear systems, resulting in truncation and cancellation errors or simply providing wrong results. In particular by applying the Taylor expansion to the second-order centered difference quotient we derive a machine precision induced approximation error of $\frac{\epsilon}{h^2}$, with ϵ being the rounding error.

AD [6] solves this problem analytically, changing the underlying code to compute derivatives by applying symbolic differentiation rules to individual assignments and using the chain rule to propagate derivatives along the flow of control. The achieved accuracy only depends on the machine's precision ϵ . There exist two distinct derivative models, differing in the order of application of the associative chain rule. Let ∇F be the Jacobian of F . The *tangent-linear* code

$$F(\overset{\downarrow}{\mathbf{x}}, \overset{\downarrow}{\mathbf{y}}) \xrightarrow{\text{dcc}} \overset{\downarrow}{F}(\overset{\downarrow}{\mathbf{x}}, \overset{\downarrow}{\mathbf{x}}, \overset{\downarrow}{\mathbf{y}}, \overset{\downarrow}{\dot{\mathbf{y}}}), \quad (6)$$

where

$$\dot{\mathbf{y}} = \nabla F(\mathbf{x}) \cdot \dot{\mathbf{x}}$$

and $\mathbf{y} = F(\mathbf{x})$,

of F computes the directional derivative $\dot{\mathbf{y}}$ of the outputs \mathbf{y} with respect to the inputs \mathbf{x} for a given direction $\dot{\mathbf{x}} \in \mathbb{R}^n$, while arrows designate inputs and outputs. By iteratively setting $\dot{\mathbf{x}}$ equal to each of the n Cartesian basis vectors in \mathbb{R}^n , we accumulate the entire Jacobian. This leads to a runtime complexity identical to finite differences of $\mathcal{O}(n) \cdot \text{cost}(F)$, where $\text{cost}(F)$ denotes the computational cost of a single function evaluation.

By exploiting the associativity of the chain rule, the *adjoint* code

$$F(\overset{\downarrow}{\mathbf{x}}, \overset{\downarrow}{\mathbf{y}}) \xrightarrow{\text{dcc}} \overset{\downarrow}{\bar{F}}(\overset{\downarrow}{\mathbf{x}}, \overset{\downarrow}{\mathbf{x}}, \overset{\downarrow}{\mathbf{y}}, \overset{\downarrow}{\bar{\mathbf{y}}}), \quad (7)$$

where

$$\mathbf{y} = F(\mathbf{x})$$

and $\bar{\mathbf{x}} = \bar{\mathbf{x}} + \nabla F(\mathbf{x})^T \cdot \bar{\mathbf{y}}$,

of F computes *adjoints* $\bar{\mathbf{x}} \in \mathbb{R}^n$ of the inputs \mathbf{x} for given adjoints $\bar{\mathbf{y}} \in \mathbb{R}^m$ of the outputs. To accumulate the entire Jacobian we have to iteratively set $\bar{\mathbf{y}}$ equal to each Cartesian basis vector of \mathbb{R}^m yielding a runtime complexity of $\mathcal{O}(m)$.

$\text{cost}(F)$. Note that for scalar functions with $m = 1$ the accumulation of the Jacobian amounts to the computation of one gradient yielding a runtime cost of $\mathcal{O}(1) \cdot \text{cost}(F)$ for the adjoint model compared to $\mathcal{O}(n) \cdot \text{cost}(F)$ for the tangent-linear model. In this particular case, we are able to compute gradients at a small constant multiple of the cost of a single function evaluation. The reduction of this factor down toward the theoretical minimum of three [6] is one of the major challenges addressed by ongoing research and development in the field of AD [7], [8].

The core idea of this paper is to develop a source transformation tool or compiler that transforms a given C code into its differentiated version. In general, this increases the differentiation order from d to $d + 1$. I.e., by taking as an input a handwritten first-order code we end up with a second-order code. Taking this insight a step further we want that our tool accepts its output as an input. Thus, starting from a given code, we are able to iteratively generate an arbitrary order of differentiation code. This unique feature is being presented in Section IV.

Furthermore our derivative code compiler is able to use checkpointing techniques for the adjoint mode, by using joint reversal as opposed to split reversal as a reversal technique. This will be explained in Section V.

III. DCC - A DERIVATIVE CODE COMPILER

Numerical optimization problems are commonly implemented as multivariate scalar functions $y = F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$, describing some residual y of a numerical model. We assume that the goal is to minimize a norm of this residual y by adapting the inputs \mathbf{x} . Therefore, for better readability and without the loss of generality, in this paper, we will only cover multivariate scalar functions.

The main link between dcc and the mathematical models of AD is the ability to decompose each function implementation into single assignment code (SAC) as follows:

$$\text{for } j = n, \dots, n + p$$

$$v_j = \varphi_j(v_i)_{i \prec j} \quad (8)$$

The entire program is regarded as a sequence of $p + 1$ elemental statements. In each statement an elemental function φ_j is applied to a set of variables $(v_i)_{i \prec j}$ yielding the unique *intermediate* variable v_j with $i \prec j$ denoting a dependence of v_j on v_i . The *independent* inputs are given by $v_i = x_i$ for $i = 0, \dots, n - 1$ while the *dependent* output of F is the final value $y = v_{n+p}$. When dcc applies the tangent-linear model to each of the $p + 1$ assignments, we obtain

$$\text{for } j = n, \dots, n + p$$

$$\dot{v}_j = \sum_{i \prec j} \frac{\partial \varphi_j}{\partial v_i} \cdot \dot{v}_i \quad (9)$$

$$v_j = \varphi_j(v_i)_{i \prec j}$$

Considering the j -th assignment in (9), the local k -th entry of the gradient $(\frac{\partial \varphi_j}{\partial v_k})_{k \prec j}$ is provided in \dot{v}_j by setting \dot{v}_k

to one and all $(\dot{v}_i)_{k \neq i \prec j}$ to zero. The gradient component $(\frac{\partial y}{\partial x_k})_{k \in \{0, \dots, n-1\}}$ is obtained by evaluating (9) and setting \dot{x}_k to one and all other $(\dot{x}_i)_{k \neq i \in \{0, \dots, n-1\}}$ to zero. To get the whole gradient we have to evaluate (9) n times letting \dot{x} range over the Cartesian basis vectors in \mathbb{R}^n . The adjoint model is acquired by transforming (8) into:

$$\begin{aligned} & \text{for } j = n, \dots, n+p \\ & \quad v_j = \varphi_j(v_i)_{i \prec j} \\ & \text{for } i \prec j \text{ and } j = n+p, \dots, n \quad (10) \\ & \quad \bar{v}_i = \bar{v}_i + \frac{\partial \varphi_j}{\partial v_i}(v_k)_{k \prec j} \cdot \bar{v}_j \end{aligned}$$

The first part consists of the original assignments $j = n, \dots, n+p$ and is called *forward section*. The *reverse section* follows with the computation of the adjoint variables in the order $j = n+p, \dots, n$. Note the reversed order of the assignments as well as the changed data flow of the left and right-hand sides compared with the original assignments. To compute the local gradient $(\frac{\partial \varphi_j}{\partial v_k})_{k \prec j}$ we have to initialize $(\bar{v}_i)_{i \prec j}$ with zero and \bar{v}_j with one. The initialization with zero is mandatory because $(\bar{v}_i)_{i \prec j}$ occurs in (10) on both sides of the adjoint assignment. According to (7), the adjoint variable \bar{v}_j is an input variable. Therefore it is initialized with the Cartesian basis vector in \mathbb{R} .

The important advantage of the adjoint model is that by evaluating (10) only once we obtain the full gradient $\frac{\partial y}{\partial \bar{x}}$ in $\bar{x}_i = \bar{v}_i$ for $i = 0, \dots, n-1$. To achieve this we have to initialize $(\bar{x}_i)_{i=0, \dots, n-1}$ with zero and \bar{y} with one. As mentioned above \bar{x} must be zero because it occurs not only on the left-hand side in (7) and y is initialized with the value of the Cartesian basis vector in \mathbb{R} .

In (8), we assumed that the input code is given as a SAC. This is an oversimplification in terms of real codes. The adjoint code has to deal with the fact that real code variables are overwritten frequently. One way to simulate the predicate of unique intermediate variables is to store certain left-hand side variables on a stack during the augmented forward section. Candidates for storing on the stack are those variables that are being overwritten and are required for later use during the computation of the local gradients and associated adjoints. Before evaluating the corresponding adjoint assignment in the reverse section the values are restored from the stack.

For illustration purposes we consider Listing 1 showing an implementation of the non-linear reduction $y(\mathbf{x}) = \prod_{i=0}^{n-1} \sin(x_i)$. `dcc` parses only functions with `void` as a return type (line 1). All inputs and return values are passed through the arguments, which in turn only consist of arrays (called by pointers) and scalar values (called by reference). Additionally we may pass an arbitrary number of integer arguments by value or by reference. We assume that all differentiable functions are implemented using values of type `double`. Therefore, only variables of type `double` are

```
1 void t1_f(int n, double* x, double* t1_x
2           , double& y, double& t1_y)
3 {
4     ...
5     for(int i=0; i<n; i++) {
6         y=y*sin(x[i]);
7         t1_y=t1_y*sin(x[i])+y*cos(x[i])*t1_x[i];
8     }
9     ...
10 }
```

Listing 2: Tangent-linear version of `f` as generated by `dcc`

```
1 for(int i=0; i<n; i++) {
2     t1_x[i]=1;
3     t1_f(n, x, t1_x, y, t1_y);
4     gradient[i]=t1_y;
5     t1_x[i]=0;
6 }
```

Listing 3: Driver for `t1_f`

directly affected by the differentiation process.

```
1 void f(int n, double *x, double &y)
2 {
3     int i=0;
4     y=0;
5     for(i=0; i<n; i++) {
6         y=y*sin(x[i]);
7     }
8 }
```

Listing 1: `dcc` input code.

Using the command line `dcc f.c -t`, we instruct the compiler to use the tangent-linear (`-t`) mode in order to generate the function `t1_f` (tangent-linear, 1st-order version of `f`) presented in Listing 2. The original function arguments `x` and `y` are augmented with their associated tangent-linear variables `t1_x` and `t1_y`. Inside a driver program this code has to be rerun n times letting the input vector `t1_x` range over the Cartesian basis vectors in \mathbb{R}^n to accumulate the entire gradient. Listing 3 shows how to use the generated code of Listing 2 in a driver program. Lines 2 and 5 let input variable `t1_x` range over the Cartesian basis vectors. By setting `t1_x[i]` to 1 the function `t1_f` (line 3) computes the partial derivative of `y` with respect to `x[i]`.

The command line `dcc f.c -a` tells `dcc` to apply the adjoint mode (`-a`) to `f.c`. The result is the function `a1_f` (adjoint, 1st-order version of `f`) shown in Listing 4. As in the tangent-linear case each function argument is augmented by an associated adjoint component, here `a1_x` and `a1_y`. As mentioned above we need a stack in the adjoint code for storing data during the forward section. The *augmented forward section* uses stacks to store values that are being overwritten and to store the control flow. The actual implementation of the stack is not under consideration here; therefore we replaced the calls to the stacks with macro definitions for better readability. By default, `dcc` generates

code that uses static arrays, which ensures high runtime performance. There are three different stacks used in the adjoint code. The stack called CS is for storing the control flow, FDS takes floating point values and IDS keeps integer values. The unique identifier of the two basic blocks [9] in the forward section are stored in lines 6 and 9. For example, after evaluating the augmented forward section of Listing 4, the stack CS contains the following sequence

$$\underbrace{0, 1, \dots, 1}_{n \text{ times}} \quad (11)$$

In line 10, variable y is stored onto the stack because it is overwritten in each iteration although needed in line 21. Hence, we restore the value of y in line 20. For the same reason we store and restore the value of i in line 11 and 19. The reverse section consist of a loop that processes the control flow stack CS. The basic block identifiers are restored from the stack and depending on the value, the corresponding adjoint basic block is executed. For example, the sequence given in (11) as content in the CS stack leads to a n -times evaluation of the adjoint basic block one and afterward one evaluation of the adjoint basic block zero. The basic block one in line 9 to 11 has the corresponding adjoint basic block in line 19 to 22. In contrast to (7), in line 22 the adjoint $a1_y$ is not incremented but assigned. This is due to the fact that y is on both hand sides of the original assignment in line 10. This brings an aliasing effect into play. This effect can be avoided with help of intermediate variables; making this code difficult to read. For that reason we show the adjoint assignment without intermediate variables. `dcc` generates adjoint assignments with intermediate variables and incrementation of the left-hand side as shown in (7). The `dcc`-generated code and the one shown here are semantically equivalent. To accumulate the gradient using the function `a1_f`, we again have to write a driver, presented in Listing 5. It is sufficient to initialize the adjoint variable `a1_y` and call the adjoint function `a1_f` only once to get the whole gradient (line 2), illustrating the reduced runtime complexity of the adjoint mode.

```

1  a1_y=1;
2  a1_f(n, x, a1_x, y, a1_y);
3  for(int j=0; j<n; j++)
4      gradient[j]=a1_x[j];

```

Listing 5: Driver for `a1_f`

IV. HIGHER ORDER DIFFERENTIATION

Numerical optimization algorithms often involve higher-order derivative models. Thus, the need for Hessians is imminent. With this in mind, `dcc` was designed to generate higher-order derivative codes effortlessly using its *reapplication feature*. `dcc` is able to generate j -th-order derivative code by reading $(j-1)$ -th-order derivative code as the input. In this section we will focus on second-order models.

```

1  void a1_f(int n, double* x, double* a1_x,
2           double& y, double& a1_y)
3  {
4      int i=0;
5      // augmented forward section
6      CS_PUSH(0);
7      y=0;
8      for ( i=0; i<n; i++) {
9          CS_PUSH(1);
10         FDS_PUSH(y); y=y*sin(x[i]);
11         IDS_PUSH(i);
12     }
13     // reverse section
14     while (CS_NON_EMPTY) {
15         if (CS_TOP==0) {
16             a1_y=0;
17         }
18         if (CS_TOP==1) {
19             IDS_POP(i);
20             FDS_POP(y);
21             a1_x[i]+=y*cos(x[i])*a1_y;
22             a1_y=sin(x[i])*a1_y;
23         }
24         CS_POP;
25     }
26 }

```

Listing 4: Adjoint `dcc` output

The tangent-linear mode reapplied to the first-order tangent-linear code (6) with $m = 1$ for scalar functions yields the second-order tangent-linear code

$$\dot{F}(\underset{\downarrow}{\tilde{\mathbf{x}}}, \underset{\downarrow}{\tilde{\mathbf{x}}}, y, \underset{\downarrow}{\tilde{y}}) \xrightarrow{\text{dcc}} \tilde{F}(\underset{\downarrow}{\tilde{\mathbf{x}}}, \underset{\downarrow}{\tilde{\mathbf{x}}}, \underset{\downarrow}{\tilde{\mathbf{x}}}, \underset{\downarrow}{\tilde{\mathbf{x}}}, y, \underset{\downarrow}{\tilde{y}}, \underset{\downarrow}{\tilde{y}}, \underset{\downarrow}{\tilde{y}}) ,$$

where

$$\begin{aligned} \tilde{y} &= (\nabla^2 F(\mathbf{x}) \cdot \tilde{\mathbf{x}})^\top \cdot \tilde{\mathbf{x}} + \nabla F(\mathbf{x}) \cdot \tilde{\mathbf{x}} , & (12) \\ \dot{y} &= \nabla F(\mathbf{x}) \cdot \tilde{\mathbf{x}} , \\ \tilde{y} &= \nabla F(\mathbf{x}) \cdot \tilde{\mathbf{x}} \quad \text{and} \\ y &= F(\mathbf{x}) . \end{aligned}$$

Again, `dcc` generates exactly the implementation of the mathematical model. As we see in (12), the term $\nabla F(\mathbf{x}) \cdot \tilde{\mathbf{x}}$ must be equal to 0 in order to accumulate the entries of the Hessian $\nabla^2 F$. As a consequence, $\tilde{\mathbf{x}}$ must be set to 0 on input. The product $(\nabla^2 F(\mathbf{x}) \cdot \tilde{\mathbf{x}})^\top \cdot \tilde{\mathbf{x}}$ represents a projection of the Hessian, determined by the vectors $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}$. In our case with $m = 1$ the Hessian $\nabla^2 F \in \mathbb{R}^{n \times n}$ has n^2 entries.

To compute the entry $\nabla F_{i,j}$ of the Hessian the vectors $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}$ have to be set to the i -th and j -th Cartesian basis vectors, respectively. In order to accumulate the whole Hessian this step has to be repeated for each entry, yielding a computational complexity of $\mathcal{O}(n^2) \cdot \text{cost}(F)$. Taking either adjoint or tangent-linear first-order input code, we reapply `dcc` by invoking `dcc -t -d 2 t1_foo.cpp`. This tells `dcc` to generate second-order (`-d 2`) tangent-linear (`-t`) derivative code while avoiding internal namespace clashes.

Looking at the possible combinations of the two differentiation models, there exist another three second-order

models. We may either apply the adjoint model to the tangent-linear code or apply the adjoint mode to the adjoint code. We will focus on the model where tangent-linear mode is applied to the adjoint code, called *tangent-linear over adjoint* mode.

This time the adjoint code (7) is taken as the input for the reapplication of the tangent-linear mode, obtaining

$$\bar{F}(\bar{\mathbf{x}}, \bar{\mathbf{x}}, y, \bar{y}) \xrightarrow{\text{dcc}} \dot{F}(\dot{\mathbf{x}}, \dot{\mathbf{x}}, \dot{\mathbf{x}}, \dot{\mathbf{x}}, y, \dot{y}, \dot{y}, \dot{y}) \quad ,$$

where

$$\begin{aligned} \dot{y} &= \nabla F(\mathbf{x}) \cdot \dot{\mathbf{x}} \quad , \\ y &= F(\mathbf{x}) \quad , \\ \dot{\mathbf{x}} &= \dot{\mathbf{x}} + \dot{\mathbf{x}}^\top \cdot \nabla^2 F(\mathbf{x}) \cdot \bar{y} + \nabla F(\mathbf{x})^\top \cdot \dot{\bar{y}} \quad \text{and} \\ \bar{\mathbf{x}} &= \bar{\mathbf{x}} + \nabla F(\mathbf{x})^\top \cdot \bar{y} \quad . \end{aligned} \quad (13)$$

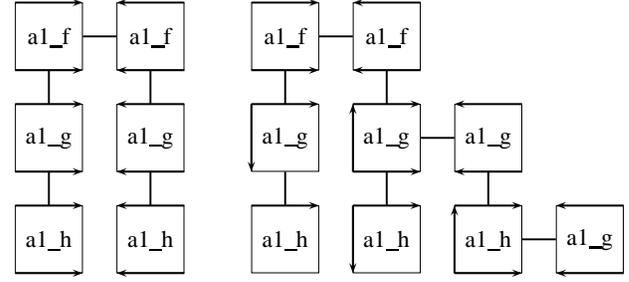
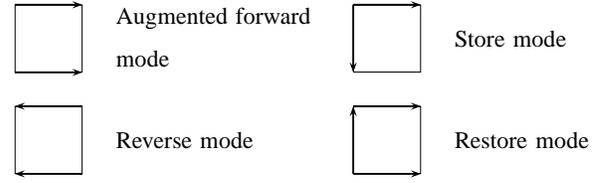
The generated implementation computes the term $\dot{\mathbf{x}}^\top \cdot \nabla^2 F(\mathbf{x}) \cdot \bar{y}$. This time we do not end up with one single entry, but we are able to harvest one complete row $\nabla^2 F_i$ of the Hessian in $\bar{\mathbf{x}}$. To achieve this, the term $\nabla F(\mathbf{x})^\top \cdot \dot{\bar{y}}$ and thus $\dot{\bar{y}}$ must be set to 0 on input. The scalar \bar{y} must be set to 1. Finally to compute a row of the Hessian $\nabla^2 F_i$, $\dot{\mathbf{x}}$ must be set to the i -th Cartesian basis vector. As such, we have to rerun this model n times in order to accumulate the whole Hessian, yielding only a linear increase in runtime complexity of $\mathcal{O}(n) \cdot \text{cost}(F)$.

The desired `dcc` command is `dcc -a -d 2 t1_foo.cpp` resulting in the file `a2_t1_foo.cpp`. The option `-a` instructs `dcc` to generate adjoint code.

V. REVERSAL STRATEGIES - CHECKPOINTING

One inherent disadvantage of the adjoint AD model over the tangent-linear model is its high memory consumption. In the reverse section of an adjoint code, each adjoint computation of a non-linear operation is dependent on a value computed during the forward run. As we have seen in Section III this value is stored on a data stack if it happens to be overwritten. In real world programs this process is not the exception but the rule. Memory locations are rewritten and reused as often as possible so that the program is as memory efficient as possible. For the adjoint AD model this results in one consumed memory location for nearly every statement. For example, updating a thousand times a variable of type double precision (e.g., $x = x + y^2$) results at least in an additional memory usage of eight thousand bytes. For each execution of this statement we have one value pushed on the stack by `FDS_PUSH(x)`.

There are several strategies to address this issue. We will present checkpointing, the core method of every AD tool to reduce memory consumption. In particular we will focus on how `dcc` deals with checkpointing and how the memory footprint may be influenced by the user.



(a) Split Reversal

(b) Joint Reversal

Figure 1: Reversal models

First we look at the reversal strategy of `dcc`. In general the adjoint model consists of a forward section and a reverse section. What happens in the case of interprocedural code where a function calls an arbitrary number of functions. There are two distinct ways of adjoining interprocedural code, namely *split reversal* and *joint reversal*.

Split reversal, presented in Figure 1a is the straightforward way of adjoining code. It strictly sticks to the adjoint model. The original code is executed in an augmented forward run. The augmentation essentially amounts to the additional stack operations introduced in Section III. These stacks are global data structures in `dcc`.

The augmented forward section is visualized by a square with two right arrows \square . One arrow stands for the values that are pushed on the stack. The other arrow represents the original function evaluation. The augmented forward section of f calls the augmented forward section of g , which itself calls the augmented forward section of h . Each function pushes its computed values on the floating point data stack (FDS).

After the augmented forward section of f the reverse section of f starts, marked by a square with two left arrows \square . This corresponds to the reverse adjoint computation with the needed function values being popped from the stack. Through the reverse section of f the reverse sections of g and h are eventually called.

In the end, there are two ways of calling a function in split reversal: in augmented forward mode and reverse mode. Memory consumption of split reversal is always directly related to the sum of pushes in the forward section.

Joint reversal, as shown in Figure 1b exploits the interprocedural structure of the program by introducing checkpoint-

ing at each function call. Each function needs to be able to store and restore its arguments.

We first start by calling f in augmented forward mode \square . If a function runs in augmented forward mode it will make subcalls in *store mode* \square . Store mode results in g storing its arguments (down arrow) and running the original code of g (right arrow), which itself calls the original code of h (right arrow).

The reverse mode of f calls g in restore mode \square . g restores its arguments and runs in augmented forward mode leading to h called in store mode. In joint reversal the forward section is immediately followed by the reverse section. So g starts its reverse section resulting in h called in restore mode. h restores its arguments and starts its forward section followed by the reverse section. After h has returned, g finished its reverse section, which eventually leads to f finalizing its reverse section.

By joining the forward and reverse section, the values that are pushed on the stack in the forward section are being popped from the stack in the following reverse section. This has two benefits. For one, memory access is structurally more local leading to a more efficient exploitation of cache memory. Additionally, memory consumption is significantly reduced since interprocedural code consumes far less memory than the sum of all the push operations. In split reversal we had two ways of calling a function whereas in joint reversal we have three; store, restore+augmented forward and reverse mode. We now compare the two reversal schemes along the call graph presented in Figure 1.

For the sake of simplicity, we assume that the original function evaluation, the forward section and the reverse section of f, g and h have each a computational cost of 1. Additionally, we assume that all the pushes of a function's forward section have a memory consumption of 1. Finally, we assume that storing the arguments of a function has no additionally memory footprint. Taking all of this into account we now compare the two reversal schemes on the call graph presented in Figure 1.

Split reversal runs all three functions in their forward and reverse section. So we end up with a computational cost of six. All the forward sections are called after each other, therefore the memory consumption is three.

In joint reversal after f has finished its forward section we have a memory consumption of 1. Only the values of f have been pushed on the stack. We assume that g is called in the middle of f . So half of the values were popped from the stack at the moment when g is called in restore mode (memory=0.5). When g ends its forward section, memory consumption is at 1.5. Assuming that h is called in the middle of g we end up with a peak memory consumption of 2 after the forward section of h . The computational cost amounts to the number of squares in the picture, which is equal to 9.

In general, joint reversal is a trade off between memory

consumption and computational cost. Memory consumption is reduced by a third from 3 to 2 whereas the computation cost has risen by fifty percent from 6 to 9. There has been more investigations into the mixing of these two strategies. [10] shows that the optimal reversal strategy is NP-complete. `dcc` uses joint reversal as its sole reversal scheme putting the emphasis on memory efficient code. In the next chapter we will demonstrate how we exploit this feature to achieve a more efficient memory footprint for our Burgers simulation.

VI. BURGERS IMPLEMENTATION

As has been described in Section I we compute the velocity field according to (2). We use dynamic programming by introducing a data array `u[i][j]` storing the velocity for a grid point i in time step j . The function h implementing the computation of the velocity field has the following signature:

```

1 void h(int& nx, // number of grid points
2       int t0, // first time step to start
3         with
4         int n, // number of time steps to
5           compute
6         double& cost, // cost function
7         double** uob, // observations
8         double** ub, // basic states
9         double** u, // model solutions
10        double* ui, // initial conditions
11        double& dx, // space increment
12        double& dt, // time increment
13        double& r, // Reynolds number
14        double& dtdx,
15        double& c0,
16        double& c1
17 )

```

Listing 6: Function h

This function computes `u[i][j]` and updates `cost` for all grid points x_i , $0 \leq i < nx$ and for all time steps $t_0 \leq j < n$. Supposing that for each time step we need do $c \cdot n_x$ pushes on the stack, we end up with approximately $c \cdot n_x \cdot n$ pushes for the entire simulation. This is also the memory consumption for calling the adjoint code `a1_h`.

The code will now be restructured according to a recursive checkpointing scheme by relying on the interprocedural joint reversal mode present in `dcc`.

```

1 void h(...) {
2     ...
3     half=n-t0/2;
4     t1=t0+half; n0=t1; n1=n;
5     if (diff > 2) {
6         g(nx, t0, n0, cost, uob, ub, u, ui, dx, dt, r,
7           dtdx, c0, c1);
8         g(nx, t1, n1, cost, uob, ub, u, ui, dx, dt, r,
9           dtdx, c0, c1);
10    }
11    else
12        h(nx, t0, n, cost, uob, ub, u, ui, dx, dt, r,
13          dtdx, c0, c1);
14 }

```

Listing 7: Function h

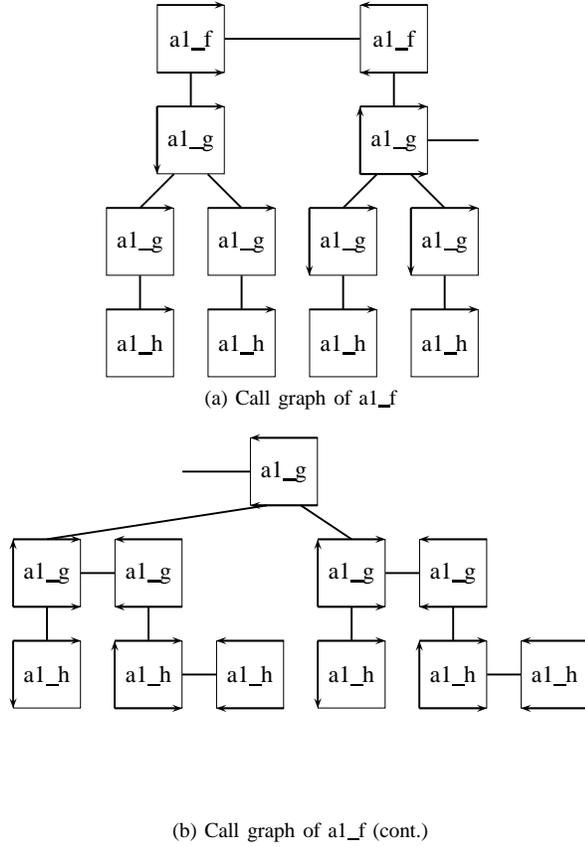


Figure 2: Burgers recursive call tree joint reversal

g has the same signature as h . Its task is to decompose the interval of time steps, calling h on a subinterval of $[t_0, n]$.

The resulting call tree as well as its joint reversed counterpart is illustrated in Figure 2.

We assume that the h has a computational cost of 1 over the entire interval from t_0 to n . If we call h in over the entire interval we end up with a forward and a reverse section adding up to a computational cost of 2. In our new structure we assume that f and g have no computational cost and no memory consumption. In our example h has a cost of $\frac{1}{2}$ since it only runs over half the interval of t_0 to n . We call h 10 times. Thus the computational cost of this call tree is 5. Note that this is independent from the depth of our recursive call tree. The memory consumption though is halved at every increase of the recursive call three depth. Ultimately memory consumption can be reduced to the number of pushes in one single time step.

VII. CASE STUDY

A. Differentiation of the original code

As discussed in Section I, we run a test case on an inverse problem based on Burgers' equation (1). As a start we take the code presented in [3] implementing the original function with the signature of

Table I: Time and memory requirements for gradient computation

n	250	500	1000	2000
f (s)	0.03	0.08	0.15	0.32
TLM (s)	33	109	457	1615
ADJ (s)	0.21	0.43	0.85	1.82
TLM-ADJ (s)	150	587	2286	8559
IDS size	7500502	15001002	30002002	60004002
FDS size	5000002	10000002	20000002	40000002
CS size	7500503	15001003	30002003	60004003

```

1 void f(int n, int nt, double& cost, double**
   u, double* ui...)
2 {
3   ...
4 }

```

Listing 8: Signature of Burgers' function

Taking n grid points of u_i as the initial conditions we integrate over nt timesteps. The values are saved in the two dimensional array u for each grid point i and time step j .

To solve the inverse problem we need the derivatives of cost with respect to the initial conditions u_i .

The results in Table I represent the runtime of one full gradient accumulation as well as the memory requirements in adjoint and tangent-linear mode. Additionally one Hessian accumulation is performed using the tangent-linear over adjoint model (13). Different problem sizes are simulated with varying n . We also mention the different stack size shown in Section III.

If we assume four bytes per integer and control stack element plus eight bytes for a floating data stack element we end up with a memory requirement of ≈ 610 MB for the Hessian accumulation. The tests were running on a GenuineIntel computer with Intel(R) Core(TM)2 Duo CPU and with 2000.000 MHz CPU.

The execution time of the tangent-linear gradient computation is growing proportionally to the problem size nx and the execution time of f :

$$FM : \frac{\text{cost}(F')}{\text{cost}(F)} \sim \mathcal{O}(n). \quad (14)$$

The single execution of tl_f takes approximately twice as long as the execution of f .

The execution time of the adjoint gradient computation is growing only proportional to the execution time of f :

$$AM : \frac{\text{cost}(F')}{\text{cost}(F)} \sim \mathcal{O}(1). \quad (15)$$

Finally we accumulate the Hessian using tangent-linear over adjoint mode. Here, the runtime is growing linearly with respect to n as well as f since the dimension of the dependent cost is equal to 1.

$$FM - AM : \frac{\text{cost}(F'')}{\text{cost}(F)} \sim \mathcal{O}(n). \quad (16)$$

Table II: Recursive checkpointing with $n = 100000$. Interval size of 100000 amounts to no recursion.

Interval size	FDS size	Runtime(s)
100000	29999610	69,3
10000	2062706	74,9
1000	433242	76,5
100	229410	79,1
10	202292	88,3

For scalar functions in particular, the runtime complexity for accumulating the Hessian using AD is the same as the runtime complexity of the gradient accumulation using finite difference. This enables developers to implement a second-order model where a first-order model has been used so far.

B. Differentiation using recursive checkpointing

Based on the recursive checkpointing scheme presented in Section V and its implementation in Section VI we conducted benchmarks varying the interval size threshold for `diff` where the recursion of `g` will stop by eventually calling `h`. The first order adjoint model was applied to compute a single gradient accumulation. The benchmarks were run on a cluster node consisting of a single thread on a Sun Enterprise T5120 cluster.

At an interval size of 100 we see major memory savings of around 98% whereas the runtime is only marginally increased by around 15% from 69,3s to 79,1s. This illustrates that checkpointing is crucial to reduce a computational problem in memory space while keeping the runtime complexity at a feasible level.

VIII. CONCLUSION & FUTURE WORK

We have presented a source transformation compiler for a restricted subset of C/C++. As such, `dcc` runs on any system with a valid C/C++ compiler making it a very portable tool. Its unique reapplication feature allows code to be transformed up to an arbitrary order of differentiation. While relying to the adjoint model for the higher-order differentiation, we save one order of magnitude of computational cost compared to a tangent-linear only or finite difference code. However, the adjoint model poses a high memory load, making an efficient checkpointing scheme crucial. Otherwise, a computation for large scale codes is even unfeasible. This is solved by resorting to interprocedural checkpointing, enabled by the joint reversal structure of the generated adjoint code. We illustrated the entire development process along a case study based on a one-dimensional implementation of the Burgers equation.

Not mentioned in this paper are several extensions not directly linked to the derivative code compiler presented here. As large simulation codes run on cluster systems, they mostly rely on parallelization techniques. The most widely used parallelization method is MPI. Hence, while applying the adjoint mode all the MPI calls need to be reversed too

[11]. This feature has been integrated into `dcc` using an adjoint MPI library [12]. Additionally there are attempts to achieve the same goal with OpenMP [13]. For the sake of brevity we also did not mention the program analysis `dcc` performs like for example *activity* and *TBR* analyses [14].

The compiler is open-source software (Eclipse Public License) and available upon request. This paper should serve as first guideline on how to differentiate C code using this tool.

Finally, the development of `dcc` is largely application driven, especially with regard to its ability in parsing the entire C/C++ language.

REFERENCES

- [1] M. Schanen, M. Foerster, B. Gendler, and U. Naumann, "Compiler-based Differentiation of Numerical Simulation Codes," in *ICCGI 2011, The Sixth International Multi-Conference on Computing in the Global Information Technology*. IARIA, 2011, pp. 105–110.
- [2] D. Zwillinger, "Handbook of Differential Equations, 3rd ed." Boston, MA, p. 130, 1997.
- [3] E. Kalnay, "Atmospheric Modeling, Data Assimilation and Predictability," 2003.
- [4] T. Kelley, *Solving Nonlinear Equations with Newton's Method*, ser. Fundamentals of Algorithms. Philadelphia, PA: SIAM, 2003.
- [5] A. Tikhonov, "On the Stability of Inverse Problems," *Dokl. Akad. Nauk SSSR*, vol. 39, no. 5, pp. 195–198, 1943.
- [6] A. Griewank and A. Walter, *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation (2nd Edition)*. Philadelphia: SIAM, 2008.
- [7] G. Corliss and A. Griewank, Eds., *Automatic Differentiation: Theory, Implementation, and Application*, ser. Proceedings Series. Philadelphia: SIAM, 1991.
- [8] G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, Eds., *Automatic Differentiation of Algorithms – From Simulation to Optimization*. New York: Springer, 2002.
- [9] A. Aho, M. Lam, R. Sethi, and J. Ullman, *Compilers. Principles, Techniques, and Tools (Second Edition)*. Reading, MA: Addison-Wesley, 2007.
- [10] U. Naumann, "DAG Reversal is NP-complete," *Journal of Discrete Algorithms*, vol. 7, no. 4, pp. 402–410, 2009.
- [11] P. Hovland and C. Bischof, "Automatic Differentiation for Message-Passing Parallel Programs," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, mar-3 apr 1998, pp. 98–104.
- [12] M. Schanen, U. Naumann, and M. Foerster, "Second-Order Adjoint Algorithmic Differentiation by Source Transformation of MPI Code," in *Recent Advances in the Message Passing Interface, Lecture Notes in Computer Science*. Springer, 2010, pp. 257–264.

- [13] M. Foerster, U. Naumann, and J. Utke, "Toward Adjoint OpenMP," RWTH Aachen, Tech. Rep. AIB-2011-13, Jul. 2011. [Online]. Available: <http://aib.informatik.rwth-aachen.de/2011/2011-13.ps.gz>
- [14] L. Hascoët, U. Naumann, and V. Pascual, "To-be-recorded Analysis in Reverse Mode Automatic Differentiation," *Future Generation Computer Systems*, vol. 21, pp. 1401–1417, 2005.

A Programming Paradigm based on Agent-Oriented Abstractions

Alessandro Ricci
University of Bologna

Via Venezia 52, 47521 Cesena (FC), Italy
a.ricci@unibo.it

Andrea Santi
University of Bologna

Via Venezia 52, 47521 Cesena (FC), Italy
a.santi@unibo.it

Abstract—More and more the notion of *agent* appears in different contexts of computer science, often with different meanings. Main ones are Artificial Intelligence (AI) and Distributed AI, where agents are exploited as a technique to develop systems exhibiting some kind of intelligent behavior. In this paper, we introduce a further perspective, shifting the focus from AI to computer programming and programming languages. In particular, we consider agents and related concepts as general-purpose abstractions useful for programming software systems in general, conceptually extending object-oriented programming with features that – we argue – are effective to tackle some main challenges of modern software development. The main contribution of the work is the definition of a conceptual space framing the basic features that characterize the agent-oriented approach as a programming paradigm, and its validation in practice by using a platform called **JaCa**, with real-world programming examples.

Keywords—agent-oriented programming; multi-agent systems; concurrent programming; distributed programming

I. INTRODUCTION

More and more the notion of *agent* appears in different contexts of computer science, often with different meanings. In the context of Artificial Intelligence (AI) or Distributed Artificial Intelligence (DAI), agents and multi-agent systems are typically exploited as a technique to tackle complex problems and develop intelligent software systems [1][2][3]. In this paper, we discuss a further perspective, which aims at exploiting the value of agents and multi-agent systems as a *programming paradigm*, providing high-levels concepts and mechanisms that are effective to tackle main challenges that characterize modern and future programming, concerning e.g. concurrency, distribution, autonomy, adaptivity.

Concurrency, in particular, due to the widespread diffusion of multi-core technologies, is more and more an important topic of mainstream programming—besides the academic research contexts where it has been studied for the last fifty years. This situation is pretty well summarized by the sentence *the free lunch is over* as put by Sutter and Larus in [4], which means that nowadays concurrency is an issue that cannot be ignored or overlooked even in everyday programming, being it more and more a must-have feature for improving performance and responsiveness of programs. Besides introducing fine-grain mechanisms or patterns to exploit parallel hardware and improve the efficiency of programs in existing mainstream lan-

guages, it is now increasingly important to introduce higher-level abstractions that “help build concurrent programs, just as object-oriented abstractions help build large component-based programs” [4]. We argue that agent-oriented programming – as framed in this paper – provides such level of abstraction, providing a rich set of concepts and mechanisms.

Actually, the idea of agent-oriented programming is not new in the context of AI/DAI: the first paper about AOP is dated 1993 [5], and since then many agent programming languages have been proposed in literature [6][7][8]. The objective of AOP as introduced in [5] was the definition of a post-OO programming paradigm for developing complex applications, providing higher-level features compared to existing paradigms. In spite of this objective, it is apparent that agent-oriented programming has not had a significant impact on mainstream research in programming languages and software development, so far. We argue that this depends on the fact that (in spite of few exceptions) most of the effort and emphasis have been put on theoretical issues related to AI themes, instead of focusing on the key principles and the practice of programming. This is the direction that we aim at exploring in our work and in this paper, which is a revised and extended version of a previous contribution [9].

The remainder of the paper is organized as follows. After presenting related work (Section II), we first define a conceptual space to describe the basic features of a general-purpose programming paradigm based on agent-oriented abstractions (Section III). Then, we provide a first practical evaluation by exploiting an agent-oriented platform called **JaCa** (Section IV), which actually integrates two different existing agent technologies, **Jason** [10] and **CARtAgO** [11]. After giving an overview of the main **JaCa** elements, in Section V we discuss in detail some selected features of **JaCa** programming, which are relevant for the development of software systems, and in Section VI we provide an overview of the application domains where **JaCa** has been effectively applied so far. Finally, in Section VII we discuss the main features that are currently missing in existing agent technologies, paving the way to the design and development of a new generation of agent-oriented programming languages. Concluding remarks are provided in Section VIII.

II. RELATED WORK ON AGENT-ORIENTED PROGRAMMING

As mentioned in the introduction, most of the agent-oriented programming languages and technologies – in particular those based on cognitive model/architectures such as the Belief-Desire-Intention (BDI) one [12] – have been introduced in the context of (Distributed) Artificial Intelligence [6][7][8]. Besides, in the context of AOSE (Agent Oriented Software Engineering) some agent-oriented *frameworks* based on mainstream programming languages – such as Java – have been introduced, targeted to the development of complex distributed software systems. A main example is JADE (Java Agent Development Framework) [13], a FIPA-compliant [14] platform that makes it possible to implement multi-agent systems in Java. JADE is based on a weak notion of agency: JADE agents are Java-based actor-like active entities, communicating by exchanging messages based on FIPA ACL (Agent Communication Language). So there is not an explicit account for high-level agent concepts – goals, beliefs, plans, intentions are examples, referring to the BDI model – that are exploited instead in agent-oriented programming languages to raise the level of abstraction adopted to define agent behaviour. Also, JADE has not an explicit notion of agent environment, defining agent actions and perceptions, which are key concepts for defining agent reactivity. Differently from JADE, the JaCa platform presented in this paper allows for programming agents using a BDI-based computational model and has an explicit notion of shared programmable environments – perceived and acted upon by agents – based on the A&A (Agents and Artifacts) conceptual model [15], described in next sections.

Another example of Java-based agent-oriented framework is *simpA* [16], which has been conceived to investigate the use of agent-oriented abstractions for simplifying the development of concurrent applications. *simpA* shares many points with the perspective depicted in this paper: however it is based on a weak model of agency, similar to the one adopted in JADE. Differently from JADE, it explicitly supports a notion of environment, based on A&A.

Besides the different underlying models, both JADE and *simpA* do not explicitly introduce a new full-fledge agent-oriented programming language for programming agents, being still based on Java. A different approach is adopted by JACK [17], a further platform for developing agent-based software, which *extends* the Java language with BDI constructs – such as goals and plans – for programming agents, integrating the object-oriented and agent-oriented levels. Finally, similarly to JADE, *Jadex* [18] is a FIPA compliant framework based on Java and XML, but adopting the BDI as underlying agent architecture.

III. AN AGENT-ORIENTED ABSTRACTION LAYER

Quoting Lieberman [19], “*The history of Object-Oriented Programming can be interpreted as a continuing quest to capture the notion of abstraction – to create computational artifacts that represent the essential nature of a situation, and to ignore irrelevant details*”. In that perspective, in

this section we identify and discuss a core set of concepts and abstractions introduced by agent-oriented programming. While most of these concepts already appeared in literature in different contexts, our aim here is to highlight their value for framing a conceptual space and an abstraction layer useful for programming complex software systems.

A. The Background Metaphor

Metaphors play a key role in computer science, as means for constructing new concepts and terminology [20]. In the case of objects in OOP, the metaphor is about real-world objects. Like physical objects, objects in OOP can have properties and states, and like social objects, they can communicate as well as respond to communications.

The inspiration for the agent-oriented abstraction layer that we discuss in this paper is anthropomorphic and refers to the A&A (Agents and Artifacts) conceptual model [15], which takes human organizations as main reference. Fig. 1 (on the left) shows an example of such metaphor, represented by a human working environment, a *bakery* in particular. It is a system where articulated concurrent and coordinated activities take place, distributed in time and space, by people working inside a common environment. Activities are explicitly targeted to some objectives. The complexity of the work calls for some division of labor, so each person is responsible for the fulfillment of one or multiple tasks. Interaction is a main dimension, due to the dependencies among the activities. Cooperation occurs by means of both direct verbal communication and through tools available in the environment (e.g., a blackboard, a clock, the task scheduler). So the environment – as the set of tools and resources used by people to work – plays a key role in performing tasks efficiently. Besides tools, the environment hosts resources that represent the co-constructed results of people work (e.g., the cake). Activity Theory [21] and distributed cognition [22] remark the fundamental role that such *artifacts* (i.e., resources and tools) have in human work and organization, both as media to enable and make it efficient communication, interaction and coordination and, more generally, to extend human cognitive and practical capabilities [23].

Following this metaphor, we see a program – or software system – as a collection of autonomous agents working and cooperating in a shared environment (Fig. 1): on the one side, agents (like humans) are used to represent and modularize those parts of the system that need some level of autonomy and pro-activity—i.e., those parts in charge to autonomously accomplish the tasks in which the overall labor is split; on the other side, the environment is used to represent and modularize the non-autonomous functionalities – called *artifacts* in Activity Theory – that can be dynamically composed, adapted and used (by the agents) to perform the tasks.

A main feature of this approach is that it promotes a *decentralized control mindset* in programming [24]. Such a mindset has two main cornerstones.

The first one is the *decentralization and encapsulation of control*: there is not a unique locus of control in the system,

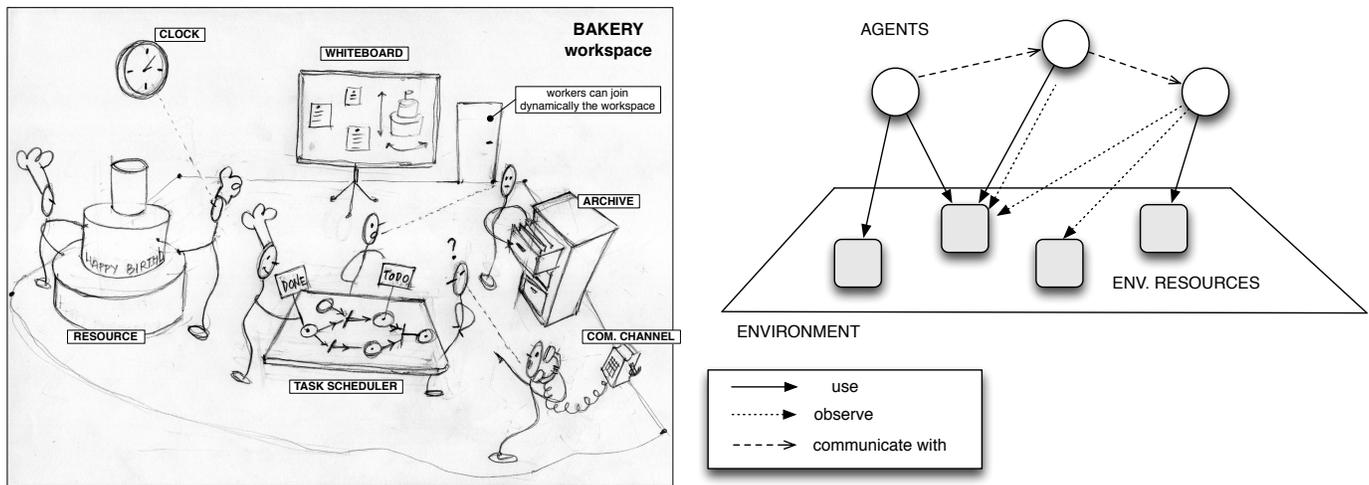


Fig. 1. (Left) Abstract representation of the A&A metaphor in the context of a bakery. (Right) Abstract representation of an agent-oriented program composed by agents working within an environment.

which is instead decentralized into agents. It is worth remarking that here we are assuming a logical point of view over decentralization—not strictly related to, for instance, physical threads or processes. The agent abstraction extends the basic encapsulation of state and behavior featured by objects by including also encapsulation of control, which is fundamental for defining and realising agent *autonomous* behaviour.

The second cornerstone is the interaction dimension, which includes coordination and cooperation. There are two basic orthogonal ways of interacting: direct communication among agents based on high-level asynchronous message passing and environment-mediated interaction (discussed in Subsection III-D) exploiting the functionalities provided by environment resources.

B. Structuring Active Behaviors: Tasks and Plans

Decentralization and encapsulation of control, as well as direct communication based on message passing, are main properties also of actors, as defined in [25]. The actor model, however, does not provide further concepts useful to *structure* the autonomous behavior, besides a simple notion of *behavior*. This is an issue as soon as we consider the development of large or simply not naive active entities. To this end, the agent abstraction extends the actor one introducing further high-level notions that can be effectively exploited to organize agent autonomous behavior, namely *tasks* and *plans*.

The notion of task is introduced to specify a unit of work that has to be executed—the objective of agents' activities. So, an agent acts in order to perform a task, which can be possibly assigned dynamically. The same agent can be able to accomplish one or more types of task, and the *type* of the agent can be strictly related to the set of task types that it is able to perform.

Conceptually, an agent is hence a computing machine that, given the description of a task to execute, it repeatedly chooses and executes *actions* so as to accomplish that task. If the task

concept is used as a way to define *what* has to be executed, the set of actions to be chosen and performed – including those to react to relevant events – represent *how* to execute such task. The first-class concept used to represent one such set is the *plan*. So the agent programmer defines the behavior of an agent by writing down the plans that the agent can dynamically combine and exploit to perform tasks. For the same task, there could be multiple plans, related to different contextual conditions that can occur at runtime.

On the one side, tasks and plans can be used to define the contract explicitly stating what jobs the agent is able to do; on the other side, they are used (by the agent programmer) to structure and modularize the description of how the agent is able to do such jobs, organizing plans in sub-plans.

This approach makes it possible to frame a smooth path in defining different levels of abstraction in specifying plans and, correspondingly, different levels of autonomy of agents. At the base level, a plan can be a detailed description of the sequence of actions to execute. In this case, task execution is fully pre-defined, since the programmer provides a complete specification of the plan; the level of autonomy of the agent is limited in selecting the plan among the possible ones specified by the programmer. In a slightly more complex case, a plan could be the description of a set of possible actions to perform, and the agent uses some criteria at runtime to select which one to execute. This enhances the level of autonomy of the agent with respect to what strictly specified by the programmer. An even stronger step towards autonomy is given by the case in which a plan is just a partial description of the possible actions to execute, and the agent dynamically infers the missing ones by exploiting information about the ongoing tasks, and about the current knowledge of its state and the state of the environment.

C. Integrating Active and Reactive Behaviours: The Agent Execution Cycle

More and more the development of applications calls for flexibly integrating active and reactive computational behaviours, an issue strongly related to the problem of integrating thread-based and event-based architectures [26]. Active behaviours are typically mapped on OS threads, and the asynchronous suspension/stopping/control of thread execution in reaction to an event is an issue in high-level languages. So, for instance, in order to make a thread of control aware of the occurrence of some event – to be suspended or stopped – it is typically necessary to “pollute” its block of statements with multiple tests spread around.

In the case of agents, this aspect is tackled quite effectively by the control architecture that governs their execution, which can be considered both *event-driven* and *task-driven*. The execution is defined by a control loop composed by a possibly non-terminating sequence of execution cycles. Conceptually, an execution cycle is composed by three different stages (see Fig. 2):

- *sense* stage – in this stage the internal state of the agent is updated with the *events* collected in the agent event queue. So this is the stage in which inputs generated by the environment during the previous execution cycle – including messages sent by the other agents – are fetched.
- *plan* stage – in this stage the next action to execute is chosen, based on the current state of the agent, the agent plans and agent ongoing tasks; additionally, agent state is also updated to reflect such a choice.
- *act* stage – in this stage the actions selected in the *plan* stage are executed.

The agent machine continuously executes these three stages, performing one execution cycle at each logical clock tick. Conceptually, the agent control flow is never blocked—actually it can be in idle state if, for instance, the executed plan states that no action has to be executed until a specific event is fetched in the sense stage. This architecture easily allows, for instance, for suspending a plan in execution and execute another plan to handle an event suddenly detected in the sense stage.

While in principle this makes an agent machine less efficient than machines without such loops, this architecture allows to have a specific point to balance efficiency and reactivity thanks to the opportunity to define proper atomic actions. Besides, in practice, by carefully design the execution cycle architecture, it is possible to minimize the overheads – for instance by avoiding to cycle and consuming CPU time if there are no actions to be executed or new events to be processed – and eventually completely avoid overheads when needed—for instance, by defining the notion of atomic (not interruptible) plan, whose execution would be as fast as normal procedures or methods in traditional imperative languages.

D. “Something is Not an Agent”: the Role of the Environment Abstraction

Often programming paradigms strive to provide a single abstraction to model every component of a system. This

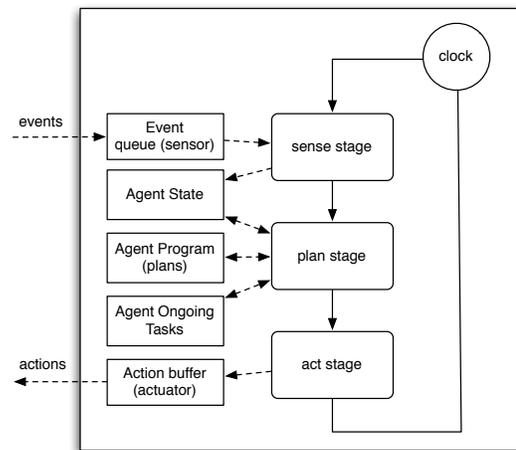


Fig. 2. Conceptual representation of an agent architecture, with in evidence the stages of the execution cycle.

happens, for instance, in the case of actor-based approaches. In Erlang [27] for example, which is actor-based, every macro-component of a concurrent system is a process, which is the actor counterpart. This has the merit of providing uniformity and simplicity, indeed. At the same time, the perspective in which everything is an active, autonomous entity is not always effective, at least from an abstraction point of view. For instance, it is not really natural to model as active entities either a shared bounded-buffer in producers/consumers architectures or a simple shared counter in concurrent programs. In traditional thread-based systems such entities are designed as monitors, which are passive.

Switching to an agent abstraction layer, there is an apparent uniformity break due to the notion of *environment*, which is a first-class concept defining the context of agent tasks, shared among multiple agents.

From a designer and programmer point of view, the environment can be suitably framed as such non-autonomous part of the system used to encapsulate and modularize those functionalities and services that are eventually shared and exploited by the autonomous agents at runtime. More specifically, by recalling the human metaphor, the environment can be framed as the set of objects functioning as *resources* and *tools* that are possibly shared and *used* by agents to execute their tasks. In order to avoid ambiguity with *objects* as defined in Object-Oriented Programming, here we will refer to these environment entities as *artifacts*, following our inspiring metaphor and adopting the terminology typically used in Activity Theory and Distributed Cognition. In that perspective, a bounded-buffer, a shared data-base etc. can be naturally designed and programmed as artifacts populating the environment where – for instance – producers/consumers agents work. Differently from agents, artifacts conceptually are not meant to be used to represent and implement autonomous / pro-active / re-active / task-oriented computational entities, but – more similar to passive objects or components or services – entities providing some functionality through a proper interface, that can be perceived and accessed by agents through actions.

E. Using and Observing the Environment

To be usable by agents, an artifact provides a set of *operations* – that constitute its *usage interface* – encapsulating some piece of functionality. Such operations are the basic actions that an agent can execute on instances of that artifact type. So the set of actions that an agent can execute inside an environment depend on the set of artifacts that are available in that environment. Since artifacts can be created and disposed at runtime by agents, the agent action repertoire can change dynamically.

The execution of an operation (action) performed by an agent on an artifact may complete with a success or a failure—so an explicit success/failure semantics is defined. Actions (operations) are performed by agents in the act stage of the execution cycle seen previously. Then, the completion of an action occurs asynchronously, and is perceived by the agent as a basic type of event, fetched in the sense stage. This can occur in the next execution cycle or in a future execution cycle, since the execution of an operation can be long-term. So, an important remark here is that the execution cycle of an agent *never blocks*, even in the case of executing actions that – to be completed – need the execution of further actions of other agents. This means that an agent, even if “waiting” for the completion of an action, can react to events perceived from the environment and execute a proper action, following what is specified in the plan.

Aside to actions, *observable properties* and *observable events* represent the other side of agent-environment interaction, that is the way in which an agent gets input information from the environment. In particular, observable properties represent the observable state that an artifact may expose, as part of its functionalities. The value of an observable property can be changed by the execution of operations of the same artifact. A simple example is a counter, providing an `inc` operation (action) and an observable state given by an observable property called `count`, holding the current count value. By observing an artifact, an agent automatically receives the updated value of its observable properties as percepts at each execution cycle, in the sense stage. Observable events represent possible signals generated by operation execution, used for making observable an information not regarding the artifact state, but regarding a dynamic condition of the artifact. Taking as a metaphor a coffee machine as an artifact, the display is an observable property, the beep emitted when the coffee is ready is an observable event. Choosing what to model as a property or as an event is a matter of environment design.

IV. EVALUATING THE IDEA WITH EXISTING AGENT TECHNOLOGIES: THE JACA PLATFORM

The aim of this section is to show more in practice some of the concepts described in the previous section. To this end, we will use existing agent technologies, in particular a platform called **JaCa**, which actually integrates two independent technologies: the **Jason** agent programming language [10] – for programming agents – and the **CARtAgO** framework [11], for programming the environment.

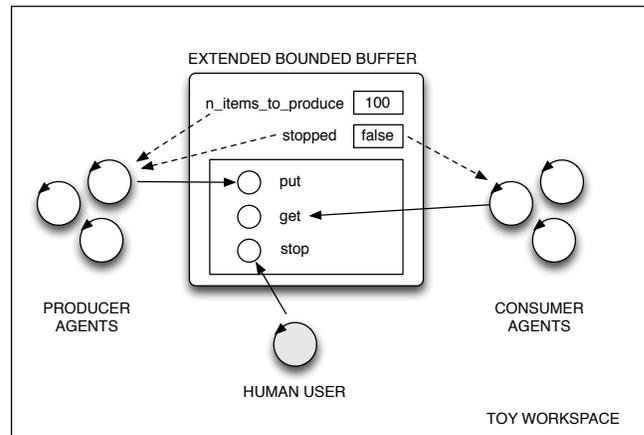


Fig. 3. A toy workspace, with producer and consumer agents interacting by means of a `ExtBBuffer` artifact.

A. JaCa Overview

Following the basic idea discussed in Section III - a **JaCa** program is conceived as a dynamic set of autonomous agents working inside a shared environment, that they use, observe, adapt according to their tasks. The environment is composed by a dynamic set of artifacts, as computational entities that agents can dynamically create and dispose, beside using and observing them.

In the following, we introduce only those basic elements of agent and environment programming that are necessary to show the features discussed at the conceptual level in the previous section. To this end, we use a toy example about the implementation of a producers-consumers architecture, where a set of producer agents continuously and concurrently produce data items that must be consumed by consumer agents (see Fig. 3). Further requirements are that (i) the number of items to be produced is fixed, but the time for producing each item (by the different producers) is not known a priori; (ii) the overall process can be interrupted by the user anytime.

The task of producing items is divided upon multiple producer agents, acting concurrently—the same holds for consumer agents. To interact and coordinate the work, agents share and use an `ExtBBuffer` artifact, which functions both as a buffer to collect items inserted by producers and to be removed by consumers and as a tool to control the overall process by the human user. The artifact provides on the one side operations (actions for the agent) to insert (`put`), remove (`get`) items and to stop the overall activities (`stop`); on the other side, it provides observable properties `n_items_to_produce` and `stopped`, keeping track of, respectively, the number of items still to be produced (which starts from an initial value and is decremented by the artifact each time a new item is inserted) and the stop flag (initially false and set to true when the `stop` operation is executed).

In the following, first we give some glances about agent programming in **Jason** by discussing the implementation of a producer agent (see Fig. 4), which must exhibit a pro-active behavior – performing cooperatively the production of items, up to the specified number – but also a reactive behavior:

if the user stops the process, the agents must interrupt their activities. For completeness, also the source code of the consumer agent is reported (Fig. 5). Then we briefly consider the implementation of the `ExtBBuffer` artifact, to show in practice some elements of environment programming.

B. Programming Agents in Jason

Being inspired by the BDI (Beliefs-Desires-Intentions) architecture [12], the Jason language constructs that programmers can use can be separated into three main categories: *beliefs*, *goals* and *plans*. An agent program is defined by an initial set of *beliefs*, representing the agent's initial knowledge about the world, a set of *goals*, which correspond to tasks as defined in Section III, and a set of *plans* that the agent can dynamically compose, instantiate and execute to achieve such goals. Logic programming is used to uniformly represent any piece of data and knowledge inside the agent program, beliefs and goals in particular.

Beliefs are represented as Prolog-like facts – that are atomic logical formulae – and represent the agent knowledge about:

- Its internal state – an example is given by the `n_items_produced(N)` belief, which is used by a producer agent to keep track of the number of items produced so far. Initially `N` is zero, and then it is dynamically updated by the agent in plans, by means of specific internal actions.
- The observable state of the artifacts that the agent is observing—in the example, every producer agent observes the `sharedBuffer` artifact, which has two observable properties: `n_items_to_produce`, representing the number of items still to be produced, and `stopped`, a flag which is set if/when the process needs to be stopped.

At design time the agent developer may want to define the agent's initial belief-base, by specifying some initial beliefs: then, beliefs can be added or removed at runtime, according to the agent changes to its state and to the resources that the agent dynamically decides to observe.

An agent program may explicitly define the agent's initial belief-base and the initial task or set of tasks that the agent has to perform, as soon as it is created. In Jason goals – i.e., tasks – are represented by Prolog atomic formulae prefixed by an exclamation mark. Referring to the example, the producer agent has an initial task to do, which is represented by the `!produce` goal. Actually, tasks can be assigned also at runtime, by sending to an agent achieve-goal messages.

Then, the main body of an agent program is given by the set of plans, which define the pro-active and reactive behavior of the agent. Agent plans are described by rules of the type `Event : Context <- Body`, where `Event` represents the specific event triggering the plan, `Context` is a boolean expression on the belief base, indicating the conditions under which the plan can be executed once it has been triggered, and `Body` specifies the sequence of actions to perform, once the plan is executed. The actions contained in a plan body can be split in three categories:

- *Internal actions*, that are actions affecting only the internal state of the agent. Examples are actions to create sub-tasks (sub-goals) to be achieved (`!g`), to manage task execution – for instance, to suspend or abort the execution of a task – to update agent inner state – such as adding a new belief (`+b`), removing beliefs (`-b`). Internal actions include also a set of primitives that allow for managing Java objects – which is the data model supported by `CARTAGO` – on the Jason side: so it is possible to create new objects (`cartago.new_obj`), invoke methods on objects (`cartago.invoke_obj`), etc.) and other related facilities (the prefix `cartago.` is used to identify in Jason the library to which the specific actions belong to).
- *External actions*, that are actions provided by the environment to interact with artifacts—as will be detailed in next section, these actions correspond to the operations provided by artifacts and included in artifact interfaces: so the repertoire of the actions of an agent is dynamic and depends on the number and type of artifacts available in the environment;
- *Communicative actions* (`.send`, `.broadcast`), which make it possible to communicate with other agents by means of message passing based on speech acts.

Referring to the example, the producer agent has a main plan (lines 8-10), which is triggered by an event `!produce` representing a new goal `!produce` to achieve. Since the agent has an initial `!produce` goal (line 4), then this plan will be triggered as soon as the agent is booted. By means of an internal action `!g`, the main plan generates two further subgoals to be achieved sequentially: `!setup` and `!produce_items`.

The plan to handle `!setup` goal (lines 12-14) creates a new instance called `sharedBuffer` of type `ExtBBuffer` by means of a predefined action called `makeArtifact`, and then starts observing it by executing the predefined action `focus` specifying its identifier. This plan fails if the artifact had been already created (by another producer), generating a `!setup` goal failure event: a plan managing the failure is specified (lines 16-18), which simply finds out the exact identifier of the existing artifact and starts observing it.

Then, two plans are specified for handling the goal `!produce_items`. One (lines 20-25) is executed if there are still items to produce—i.e., if the agent has not the belief `n_items_to_produce(0)`. Note that the value of this belief depends on the current state of the `sharedBuffer` artifact. This plan first produces a new item (subtask `!produce_item`), then inserts the item in the buffer by means of a `put` action, whose effect is to execute the `put` operation on the artifact; if this action succeeds, the plan goes on by updating the belief `n_items_produced` incrementing the number of items produced and generates a new subgoal `!produce_items` to repeat the task. Actually, when executing an external action – such as `put` – it is possible to explicitly denote the artifact providing that action, in order to avoid

```

1  /* Producer agent */
2
3  n_items_produced(0). /* initial belief */
4  !produce.             /* initial goal */
5
6  /* plans */
7
8  +!produce
9    <- !setup;
10     !produce_items.
11
12 +!setup
13   <- makeArtifact("sharedBuf", "ExtBBuffer", [], Id);
14     focus(Id).
15
16 -!setup
17   <- lookupArtifact("sharedBuf", Id);
18     focus(Id).
19
20 +!produce_items : not n_items_to_produce(0)
21   <- !produce_item(Item);
22     put(Item);
23     -n_items_produced(N);
24     +n_items_produced(N+1);
25     !produce_items.
26
27 +!produce_items : n_items_to_produce(0)
28   <- !finalize.
29
30 +!produce_item(Item) <- ...
31
32 +!finalize : n_items_produced(N)
33   <- println("completed - items produced: ", N).
34
35 -!produce_items
36   <- !finalize.
37
38 +stopped(true)
39   <- .drop_all_intentions;
40     !finalize.

```

Fig. 4. Source code of a producer agent.

ambiguities, by means of **Jason** annotations: `put(Item) [artifact_name("sharedBuffer")];`. The other plan (lines 27-28) is executed if there are no more items to produce—the `n_items_to_produce` belief referred in the plan context contains the updated value of the corresponding observable property in the artifact. In this case the `!finalize` task is executed, and it prints on standard output the number of items produced by the agent. The `println` action corresponds to the operation with the same name provided by an artifact called `console`, which is available by default in every workspace.

The reactive behavior of an agent can be realized by plans triggered by a belief addition/change/removal – corresponding to changes in the state of the environment – and by the failure of a plan in achieving some goal. In the example, the producer agent has a plan (lines 38-40) which is executed when the belief `stopped` about the observable property of the artifact is updated to `true`. This means that the user wants to interrupt and stop the production. So the plan stops and drops all the other possible plans in execution – using an internal action `.drop_all_intention` – and the `!finalize` subtask is executed.

Finally, the producer agent has also a plan (lines 35-36) to react to the failure of the `!produce_items` task, which is expressed by the event `-!produce_items`. This can

```

1  /* Consumer agent */
2
3  !consume.
4
5  +!consume: true
6    <- ?bufferReady;
7      !consumeItems.
8
9  +!consumeItems
10   <- get(Item);
11     !consumeItem(Item);
12     !consumeItems.
13
14 +!consumeItem(Item) <- ...
15
16 +?bufferReady : true
17   <- lookupArtifact("sharedBuffer", _).
18
19 -?bufferReady : true
20   <- .wait(50);
21     ?bufferReady.

```

Fig. 5. Source code of a consumer agent.

```

1  /* Main of the multi-agent program */
2
3  MAS prodcons {
4    environment: c4jason.CartagoEnvironment
5
6    agents:
7      producer agentArchClass c4jason.CAgentArch #10;
8      consumer agentArchClass c4jason.CAgentArch #10;
9  }

```

Fig. 6. Main configuration file of the producers-consumers program.

happen when the agent, believing that there are still items to be produced, starts the plan to produce a new item and tries to insert it in the buffer. However, the `put` action fails because other agents produced in the meanwhile the missing items.

The semantics of the execution of plans reacting to events is defined by **Jason** reasoning cycle [10] (shown in Fig. 7), which is a more articulated version of the execution cycle described in Section III. In particular, the plan stage in this case includes multiple steps, to select – given an event – a plan to be executed. So an agent can have multiple plans in execution but only one action at a time is selected (in the plan stage) and executed (in the act stage). By executing an action, a plan is suspended until the action is completed (with success or failure). A detailed description of the cycle – as well as of the **Jason** syntax – can be found in [10].

C. Programming the Environment in *CARTAgO*

The implementation of the `ExtBBuffer` artifact is shown in Fig. 8. Being **CARTAgO** a framework on top of the Java platform, artifact-based environments can be implemented using a Java-based API, exploiting the annotation framework. Here we don't go too deeply into the details of such API, we just introduce the main concepts that have been mentioned in Section III; for more information, the interested reader can refer to **CARTAgO** papers [11] and the documents that are part of **CARTAgO** distribution [28].

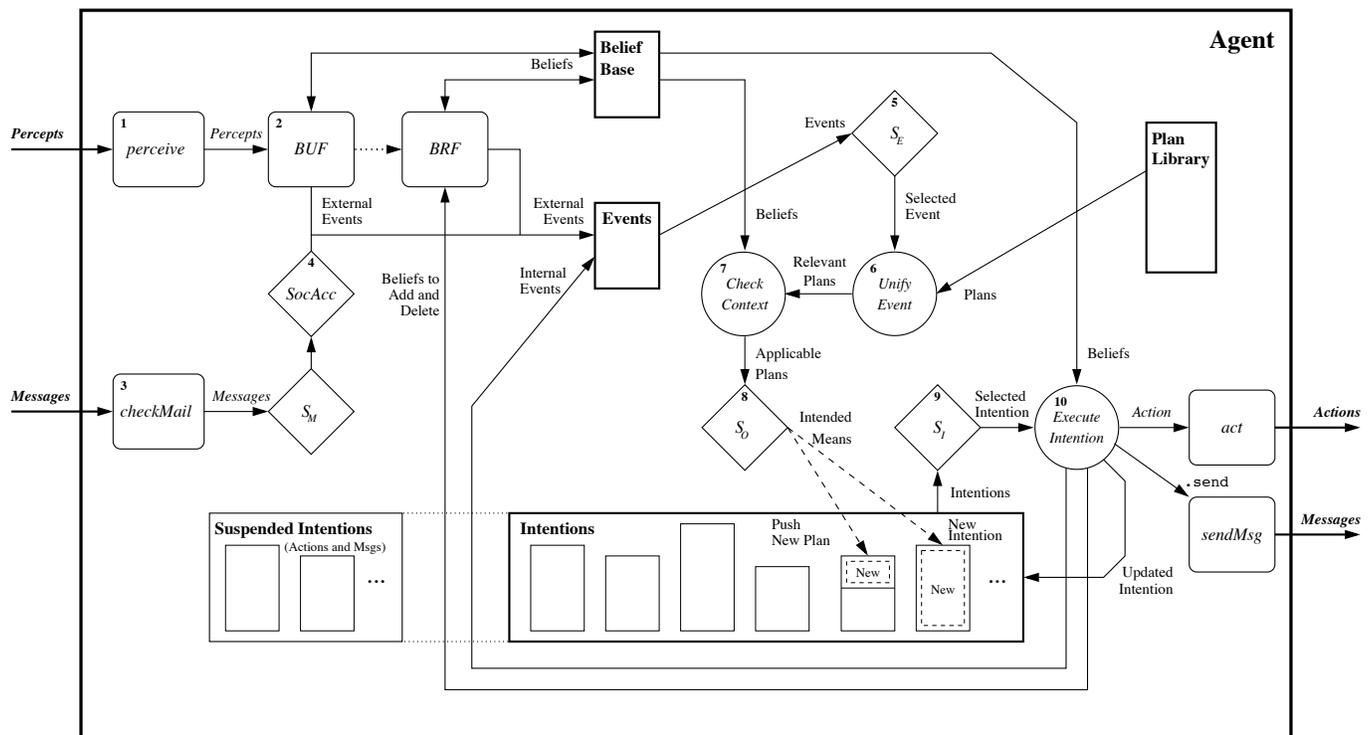


Fig. 7. A representation of JASON reasoning cycle (taken from [10]). In the first steps, the environment is perceived (step 1) and the beliefs about the state of the environment updated (step 2), by means of a customizable belief-update function (BUF). Besides input information from the environment, beliefs are updated also with messages possibly sent by other agents (step 3), filtered according some criteria defining “socially acceptable” messages (step 4). Then, updates to the belief base generate external events, appended in the event queue. This concludes the *sense* stage. Then, in the *plan* stage events are considered one by one (step 5), and for each one a relevant and applicable plan is selected (step 5 and 6), if available, from the plan library. If a plan is found, a new intention is instantiated (step 8), representing the plan in execution. The plan stage is completed by selecting the next action to do from one of the ongoing intentions (step 9). Finally, in the *act* stage the selected action is executed (step 10), and the cycle starts again.

In CARTAgO, an artifact type can be defined by extending a base *Artifact* class. Artifacts are characterized by a usage interface containing a set of operations that agents can execute to get some functionalities. In the example, the artifact *ExtBBBuffer* provides three operations: *put*, *get* and *stop*. The *put* operation inserts a new element in the buffer – decrementing the number of items to be produced – if the stopped flag has not been set, otherwise the operation (action) fails. The *get* operation removes an item from the buffer, returning it as a feedback of the action. The *stop* operation sets the *stopped* observable property to true.

Operations are implemented by methods annotated with `@OPERATION`. The *init* method is used as constructor of the artifact, getting the initial parameters and setting up the initial artifact state. Inside an operation, guards can be specified (*await* primitive), which suspend the execution of the operation until the specified condition over the artifact state (represented by a boolean method annotated with `@GUARD`) holds. In the example, the *put* operation can be completed only when the buffer is not full (*bufferNotFull* guard) and the *get* one when the buffer is not empty (*bufferNotEmpty* guard). The execution of operations inside an artifact is *transactional*: among the other things, this implies that at runtime multiple operations can be invoked concurrently on an artifact but only

one operation can be in execution at a time—the other ones are suspended. On the agent side, when executing an external action, the agent plan is suspended until the corresponding artifact operation has completed (i.e., the action completed). Then, the action succeeds or fails when (if) the corresponding operation has completed with success or failure.

Besides operations, artifacts typically have also a set of observable properties (*n_items_to_produce* and *stopped* in the example), as data items that can be perceived by agents as environment state variables. Instance fields of the class instead are used to implement the non observable state of the artifact—for instance, the list of items *items* in the example. Observable properties can be defined, typically during artifact initialization, by means of the *defineObsProperty* primitive, specifying the property name and initial value (lines 11-12). Inside operations, observable properties value can be inspected and changed dynamically by means of primitives such as: *getObsProperty*, to retrieve the current value of an observable property (see, for instance, lines 18 and 22), *updateObsProperty* to update the value, or *updateValue* on an *ObsProperty* object, once the property has been retrieved with *getObsProperty* (line 23).

Besides observable properties, an artifact can make it ob-

```

1  import cartago.*;
2
3  public class ExtBBuffer extends Artifact {
4
5      private LinkedList<Object> items;
6      private int bufSize;
7
8      void init(int bufSize, int nItemsToProd){
9          items = new LinkedList<Object>();
10         this.bufSize = bufSize;
11         defineObsProperty("n_item_to_produce",nItemsToProd);
12         defineObsProperty("stopped", false);
13     }
14
15     @OPERATION void put(Object obj){
16         await("bufferNotFull");
17         ArtifactObsProperty stopped =
18             getObsProperty("stopped");
19         if (!stopped.booleanValue()){
20             items.add(obj);
21             ArtifactObsProperty p =
22                 getObsProperty("n_item_to_produce");
23             p.updateValue(p.intValue() - 1);
24         } else {
25             failed("no_more_items_to_produce");
26         }
27     }
28
29     @GUARD boolean bufferNotFull(){
30         return items.size() < nmax;
31     }
32
33     @OPERATION void get(OpFeedbackParam<Object> result){
34         await("itemAvailable");
35         Object item = items.removeFirst();
36         result.set(item);
37     }
38
39     @GUARD boolean itemAvailable(){
40         return items.size() > 0;
41     }
42
43     @OPERATION void stop(){
44         updateObsProperty("stopped",true);
45     }
46 }

```

Fig. 8. Source code of the ExtBBuffer artifact.

servable also events occurring when executing operations. This can be done by using a signal primitive, specifying the type of the event and a list of actual parameters. For instance, `signal("my_event", "test", 0)` generates an observable event `my_event("test", 0)`. In the example, to notify the stop we could generate a `stopped` signal in the `stop` operation, instead of using an observable property. Observable events are perceived by all agents observing the artifact—which could react to them as in the case of observable property change.

Java objects and primitive data types are used as data model binding the agent and artifact layers, in particular to encode parameters in operations, fields in observable properties and signals.

To summarize, operations are computational processes occurring inside the artifact, possibly changing the observable properties and generating signals that are relevant for the agents using/observing the artifact. An operation is executed as soon as an agent triggers its execution – by executing the corresponding action. Given the transactional execution semantics adopted, only one operation can be in execution at a

time—so no interferences and race conditions occur if multiple agents use concurrently the same artifact. Like in the case of monitors, other operations that are possibly concurrently triggered are blocked (suspended). The conditions that can be specified with the `await` command are conceptually similar to condition variables. Differently from the monitor case (with threads or processes), if an operation (action) is suspended, the agent that executed it is not: the execution cycle goes on, to eventually react to percepts and/or select and execute other actions from other plans.

Other features of the artifact model implemented in CArTAgO include: (i) the capability of *linking* together artifacts, making it possible for an artifact to execute operations (called linked operations) on other artifacts; (ii) the capability of triggering the execution of *internal* operations from other operations of the same artifact; and (iii) the capability of specifying for each artifact type a *manual*, i.e., a machine readable document containing the description of the functionalities provided by the artifacts of this type and the operating instructions, i.e., how to exploit such functionalities.

D. The Multi-Agent Program in the Overall

Finally, the *main* or entry point of a JaCa multi-agent program is given by a Jason source file – with extension `*.mas2j` – describing the initial configuration of the system, in particular the name of the MAS and the initial set of the agents that must be created and possibly some information and attributes that concern environment and agent implementation. The configuration file for the example is shown in Fig. 6, where ten instances of `producer` agents and ten instances of `consumer` agents are spawned. To launch multiple agents of the same type (e.g., ten producer agents) the cardinality can be specified as a parameter in the declaration (#10); the unique name of the agent in this case is given by the type and a progressive integer (in the example: `producer1`, `producer2`, etc).

By default, a single workspace called `default` is created and the specified agents are joined to this workspace. Actually a JaCa program can be composed by multiple workspaces and agents can concurrently join and work in multiple workspaces, either locally or in remote JaCa nodes. Workspaces can be created dynamically by agents by exploiting functionalities that are provided by a set of artifacts that are available, by default, in each workspace. Among the others, such a set includes: a `console` artifact, providing functionalities for printing on standard output; a `workspace` artifact, providing functionalities for managing the current workspace, including creating new artifacts (`makeArtifact` operation), disposing existing artifacts (`disposeArtifact`), discovering the identifier of existing artifacts (`lookupArtifact`), setting the security policies ruling the agent access to artifacts, etc.; a `blackboard` artifact, functioning as a blackboard – or better as a tuple space [29] – providing functionalities for enabling indirect communication and coordination among agents.

V. JACa PROGRAMMING: FURTHER FEATURES

In this section we focus on three main programming features among the others that are provided by JaCa, namely the capability of exploiting both direct communication based on message passing and indirect interaction through artifacts, the support for building distributed programs and the capability of integrating existing libraries, such as GUI toolkits. Further features are described in JaCa and CArtaGo technical documentation.

A. Integrating Direct Communication and Mediated Interaction

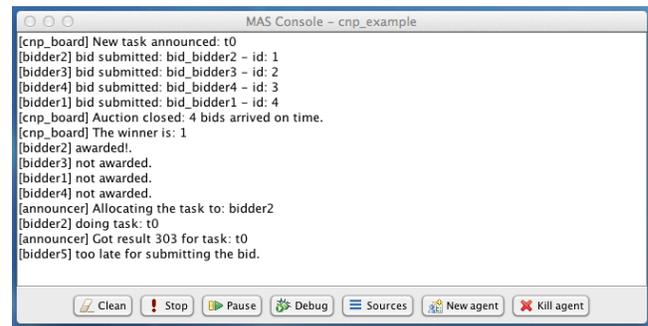
In JaCa agents can interact and communicate in two basic ways, either exchanging messages through speech acts [30] or by sharing and co-using artifacts functioning as interaction and coordination media [31]. The first way is generally referred as direct communication, while the latter as indirect or mediated communication. Both types of communication are important in programming concurrent and distributed programs, and we allow for exploiting them together.

The direct communication model is the one provided by the JASON language, based on a comprehensive subset of the KQML Agent Communication Language [30]. Among the available performatives, `tell` makes it possible to inform the receiver agent about some information (stored in the target agent as a belief), `achieve` to assign a new goal, and `ask` to request information. These performatives must be included in the communication action (`.send`) that actually sends the message, along with the specific parameters. An agent can react to the arrival of messages or, at a higher level, to the effect that the speech acts have, that are uniformly modeled as belief addition (for the `tell` performative) or goal addition (for the `achieve` performative).

To give a concrete taste of the approach, in the following we describe the realization of simplified version of the Contract Net Protocol (CNP) [32], in which both direct message passing and artifacts are used. In the example, a `ContractNetBoard` artifact (Fig. 11) is used by an announcer agent (code shown in Fig. 9) and five bidder agents (Fig. 10) to help their coordination in choosing the agent to whom allocate a task `todo`. Once the agent has been chosen, direct communication is used between the allocator of the task and the chosen agent to allocate the task and get the results.

Some brief explanation of the program behavior. In the main configuration file (Fig. 12), one announcer agent and five bidder agents are launched. The announcer opens the auction to allocate the task by performing an `announce` action over the `cnp_board` artifact (line 7). The artifact is observed and used also by the bidder agents, who are available for doing tasks. The `announce` action/operation executed by the announcer creates a new observable property `task_todo`, storing information about the new task (Fig. 11, lines 12-17).

As soon as a bidder perceives that there is a new task to do, it reacts (Fig. 10, lines 10-22) by computing a new bid and issuing them on the contract net board by performing a `bid` action. The action can fail if the auction has been already



```

MAS Console - cnp_example
[cnp_board] New task announced: t0
[bidder2] bid submitted: bid_bidder2 - id: 1
[bidder3] bid submitted: bid_bidder3 - id: 2
[bidder4] bid submitted: bid_bidder4 - id: 3
[bidder1] bid submitted: bid_bidder1 - id: 4
[cnp_board] Auction closed: 4 bids arrived on time.
[cnp_board] The winner is: 1
[bidder2] awarded!
[bidder3] not awarded.
[bidder1] not awarded.
[bidder4] not awarded.
[announcer] Allocating the task to: bidder2
[bidder2] doing task: t0
[announcer] Got result 303 for task: t0
[bidder5] too late for submitting the bid.
  
```

Fig. 13. An execution trace of the CNP program, displayed on the Jason console.

closed by the announcer: in that case a message is printed on the console (lines 20-22). On the artifact side, the `bid` operation (lines 19-28) just adds the new bid to the list of bids received so far, if the auction is still opened, otherwise the operation fails (by executing the `failed` artifact primitive). As a detail, the third parameter of the `bid` operation is an action feedback parameter, i.e., an output parameter of the action bound to some value by the operation execution itself, set with a fresh identifier univocally identifying the bid.

The announcer waits some amount of time (2 seconds in the example), and then closes the auction by invoking the `close` operation (lines 8-9), which results in changing the state observable property of the artifact to "closed" and returns the list of information about the received bids as an action feedback parameter (lines 30-36). Such information are represented by instances of the `Bid` class. Then, the agent selects a bid (in the example the first one) and awards the bidder by performing an `award` action (lines 10-11), which results in updating the content of the `winner` observable property in the artifact (lines 38-41).

This change is perceived by bidder agents, which react in a different way depending on the fact that they are the winner or not (lines 24-28). After awarding, the announcer then communicates directly with the winner bidder by sending an `achieve` message specifying the task to be done (line 15). To retrieve the identifier of the bidder agent to whom sending the message, the method `getWho` is invoked on the selected bid object by means of the `cartago.invoke_obj` internal action.

Then, the awarded bidder reacts to the new goal to achieve (lines 35-37), just printing a message and then sending a message to inform the announcer about the task result (line 37). Finally the announcer reacts to the new belief communicated by the bidder (lines 19-20) by printing the result on the console.

A possible execution trace that can be obtained by launching the program is reported in Fig. 13, which shows the content of the JASON console. In that specific execution, four bidders were able to submit their bid on time and the winner was the bidder `bidder2` (whose bid identifier assigned by the `cnp_board` was 1).

```

1  /* announcer agent */
2
3  !allocate_task("t0",2000).
4
5  +!allocate_task(Task,Deadline)
6    <- makeArtifact("cnp_board","ContractNetBoard",[]);
7    announce(Task);
8    .wait(Deadline);
9    close(Bids);
10   !select_bid(Bids,Bid);
11   award(Bid);
12   cartago.invoke_obj(Bid,getWho,Who);
13   println("Allocating the task to: ",Who);
14   .my_name(Me);
15   .send(Who,achieve,task_done(Task,Me)).
16
17 +!select_bid([Bid|_],Bid).
18
19 +task_result(Task,Result)
20   <- println("Got result ",Result," for task: ",Task).

```

Fig. 9. Source code of the announcer agent.

```

1  /* bidder agent */
2
3  task_result("t0",303).
4  !look_for_tasks("t0").
5
6  +!look_for_tasks(Task)
7    <- +task_descr(Task);
8      focusWhenAvailable("cnp_board").
9
10 +task_todo(Task) : task_descr(Task)
11   <- !make_bid(Task).
12
13 +!make_bid(Task)
14   <- !create_bid(Task,Bid);
15     .my_name(Me);
16     bid(Bid,Me,BidId);
17     +my_bid(BidId);
18     println("Bid submitted: ",Bid," - id: ",BidId).
19
20 -!make_bid(Task)
21   <- println("Too late for submitting the bid.");
22     .drop_all_intentions.
23
24 +winner(BidId) : my_bid(BidId)
25   <- println("awarded!.").
26
27 +winner(BidId) : my_bid(X) & not my_bid(BidId)
28   <- println("not awarded.").
29
30 +!create_bid(Task,Bid)
31   <- .wait(math.random(3000));
32     .my_name(Name);
33     .concat("bid_",Name,Bid).
34
35 +!task_done(Task,ResultReceiver): task_result(Task,Res)
36   <- println("doing task: ",Task);
37     .send(ResultReceiver,tell,task_result(Task,303)).

```

Fig. 10. Source code of bidder agents.

```

1  /* Contract Net Board artifact */
2
3  public class ContractNetBoard extends Artifact {
4    private List<Bid> bids;
5    private int bidId;
6
7    void init(){
8      this.defineObsProperty("state","closed");
9      bids = new ArrayList<Bid>();
10   }
11
12   @OPERATION void announce(String taskDescr){
13     defineObsProperty("task_todo", taskDescr);
14     getObsProperty("state").updateValue("open");
15     bids.clear(); bidId = 0;
16     log("New task announced: "+taskDescr);
17   }
18
19   @OPERATION void bid(String bid, String who,
20     OpFeedbackParam<Integer> id){
21     if (getObsProperty("state").stringValue().equals("open")){
22       bidId++;
23       bids.add(new Bid(bidId,who,bid));
24       id.set(bidId);
25     } else {
26       this.failed("cnp_closed");
27     }
28   }
29
30   @OPERATION void close(OpFeedbackParam<Bid[]> bidList){
31     getObsProperty("state").updateValue("closed");
32     int nbids = bids.size();
33     Bid[] vect = new Bid[nbids]; bids.toArray(vect);
34     bidList.set(vect);
35     log("Auction closed: "+nbids+" bids arrived on time.");
36   }
37
38   @OPERATION void award(Bid prop){
39     signal("winner", prop.getId());
40     log("The winner is: "+prop.getId());
41   }
42
43   static public class Bid {
44     private int id;
45     private String who, descr;
46
47     public Bid(int id, String who, String descr){
48       this.descr = descr; this.id = id; this.who = who;
49     }
50     public String getWho(){ return who; }
51     public int getId(){ return id; }
52     public String getDescr(){ return descr; }
53     public String toString(){ return descr; }
54   }
55 }

```

Fig. 11. Source code of the CNP board artifact.

```

1  MAS cnp_example {
2    environment: c4jason.CartagoEnvironment
3    agents:
4      announcer agentArchClass c4jason.CAgentArch;
5      bidder agentArchClass c4jason.CAgentArch #5;
6  }

```

Fig. 12. Main configuration file of the CNP example, spawning one announcer agent and five bidder agents.

B. Distributed Programming and Open Systems Programming

JaCa intrinsically supports concurrent programming, in different ways: by exploiting JASON runtime architecture, agents are executed concurrently (and in parallel on a parallel HW, such as multi-core architectures); also, artifacts are executed concurrently, that is operations requested on different artifacts are executed concurrently.

Besides, JaCa directly supports also distributed programming: an agent running on some node can join workspaces that are hosted on a remote nodes, and then work with artifacts of the remote workspace(s) transparently. A simple example is shown in Fig. 14, in which an agent joins a remote `test` workspace located in `acme.org`, and, there, the agent prints some information on the console, creates a new `Counter` artifact called `c0` and uses it, by executing the `inc` operation and reacting to changes to the `count` observable property. While working on multiple workspaces, in JaCa a notion of *current* workspace is defined, being it the workspace implicitly referred when the agent invokes an operation over an artifact without specifying its full identifier. `current_wsp` is a predefined agent belief keeping track of current workspace. When an agent starts its execution, the current workspace is set by default to the `default` workspace. Then, it is automatically updated as soon as the agent joins other workspaces (including remote ones) or the agent executes a predefined `set_current_wsp` action. So, in the example, by joining the remote `test` workspace, this becomes the current workspace, and then the `println` action acts on the `console` artifact there, as well as the `makeArtifact` action that creates a new artifact overthere too. It is worth noting that in the plan reacting to a change to the `count` observable property (mapped on `count` belief), the agent prints a message on the console in the *original* workspace (lines 19-21): to disambiguate what console to use, in the action an annotation reporting the workspace where the artifact is stored is specified (line 21). The agent source code includes also a plan reacting to a failure in the plan handling the `!use_remote` goal, due to the fact that a `Counter` artifact called `c0` was already present in the remote workspace.

So in the overall this facility makes it possible to implement open systems with dynamic and distributed structure and behavior, given by the capability of agents of spawning other new agents dynamically, of joining dynamically existing workspaces or creating new ones, of creating / disposing artifacts belonging to a workspace. Given the distributed programming facility, a workspace can be joined by unknown agents of JaCa programs that have been spawned independently from the program where the workspace has been defined. The possibility of explicitly specifying security policies at a workspace level – by exploiting the functionalities provided by the `workspace` artifact – makes it possible to rule and govern such openness according to the need.

C. Wrapping Existing Libraries and External Resources

Specific kind of artifacts can be designed and used to wrap and reuse existing libraries – written in Java but also in other

```

1 !test_remote.
2
3 +!test_remote
4   <- ?current_wsp(Id,_,_)
5     +default_wsp(Id);
6     println("testing remote..");
7     joinRemoteWorkspace("test","acme.org",WspID2);
8     ?current_wsp(_,WName,_);
9     println("hello there ",WName);
10    !use_remote;
11    quitWorkspace.
12
13 +!use_remote
14   <- makeArtifact("c0","examples.Counter",[],Id);
15     focus(Id);
16     inc;
17     inc.
18
19 +count(V)
20   <- ?default_wsp(Id);
21     println("count changed: ",V)[wsp_id(Id)].
22
23 -!use_remote
24   [makeArtifactFailure("artifact_already_present",_)
25   <- ?default_wsp(WId);
26     println("artifact already created ")[wsp_id(WId)];
27     lookupArtifact("c0",Id);
28     focus(Id);
29     inc.

```

```

1 public class Counter extends Artifact {
2
3   void init(){
4     defineObsProperty("count",0);
5   }
6
7   @OPERATION void inc(){
8     ObsProperty prop = getObsProperty("count");
9     prop.updateValue(prop.intValue()+1);
10  }
11 }

```

Fig. 14. An agent joining and working in a remote workspace (*top*), and the source code of the counter used and observed remotely (*bottom*).

languages, such as C and C++, exploiting the JNI (Java Native Interface) mechanism – making their functionalities available to agents, with a clean and uniform interface—which is the one provided by the artifact model. This allows in particular to build JaCa libraries that make it possible to access and interact with external resources existing in the deployment context or outside the system (such as a Web Services, a database, a legacy system).

A main example of JaCa library wrapping and integrating existing technologies is the one that allows for building and exploiting graphical user interface (GUI) toolkits. GUIs inside a JaCa program are modeled as artifacts mediating the interaction between humans and agents. A basic abstract artifact `GUIArtifact` is provided to be extended in order to create concrete GUIs. A GUI is designed then to make it observable to interested agents the events generated by the components (buttons, edit fields, list boxes,...) inside the GUI. Also, as an artifact, it provides operations that allow agents to interact with the GUI themselves, for instance to set the content of text fields.

Fig. 15 shows a simple example, in which an agent uses a GUI to repeatedly display the output of its work and to promptly react to user input. In particular, the agent creates

```

1 package c4jexamples;
2 ...
3 public class View extends GUIArtifact {
4     private MyFrame frame;
5
6     public void setup() {
7         frame = new MyFrame();
8         defineObsProperty("value",0);
9         linkActionEventToOp(frame.stopButton,"stop");
10        linkWindowClosingEventToOp(frame, "close");
11        frame.setVisible(true);
12    }
13
14    @INTERNAL_OPERATION void stop(ActionEvent ev){
15        signal("stopped");
16    }
17
18    @INTERNAL_OPERATION void close(WindowEvent ev){
19        signal("closed");
20    }
21
22    @OPERATION void setOutput(int value){
23        frame.updateOutput(""+value);
24        getObsProperty("value").updateValue(value);
25    }
26
27    class MyFrame extends JFrame {
28        private JButton stopButton;
29        private JTextField output;
30
31        public MyFrame(){
32            setTitle(":: View ::");
33            setSize(200,100);
34            JPanel panel = new JPanel();
35            setContentPane(panel);
36            stopButton = new JButton("stop");
37            stopButton.setSize(80,50);
38            output = new JTextField(10);
39            output.setText("0"); output.setEditable(true);
40            panel.add(output); panel.add(stopButton);
41        }
42        public void updateOutput(String s){
43            output.setText(s);
44        }
45    }
46 }

```

```

1 count(0).
2 !do_task_with_view.
3
4 +!do_task_with_view
5     <- makeArtifact("gui","c4jexamples.View",[],Id);
6         focus(Id);
7         !do_task.
8
9 +!do_task
10     <- -count(C);
11         C1 = C + 1;
12         +count(C1);
13         setOutput(C1);
14         !do_task.
15
16 +stopped : value(V)
17     <- .drop_all_intentions;
18         println("stopped - value: ",V).
19
20 +closed
21     <- .my_name(Me);
22         .kill_agent(Me).

```

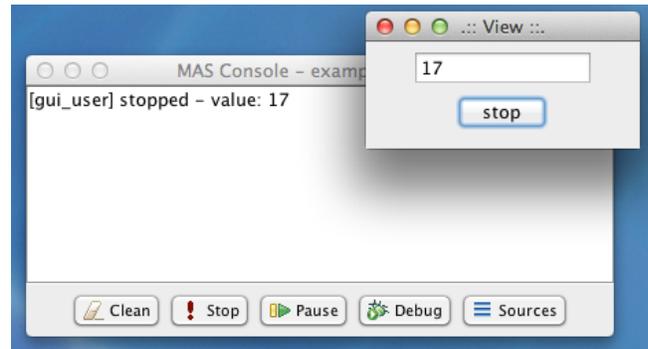


Fig. 15. Implementing and using GUI in JaCa: the View artifact (left), the agent using the GUI (right—top) and the output of the program (right—bottom).

a GUI artifact called *View*, providing one *stop* button and one output edit text. The structure of the GUI – based on Java Swing library – is defined by the *MyFrame* class, as it would be in a traditional OO program. An instance of this class is created inside *View* and events generated by the GUI components are linked to internal operations of the artifact by means of a set of predefined methods implemented in *GUIArtifact*. In particular an action event generated by `frame.stopButton` causes the execution of the internal operation *stop*, which generates an observable event *stopped*, and the window closing event is mapped onto the *close* operation, which generates a *closed* event. The agent first creates an instance called *gui* of the *View* artifact, and then repeatedly uses the view to display the results of its task, by means of the *setOutput* action (operation). While doing this task, the agent also observes the GUI and as soon as a *stopped* event is perceived, the agent reacts by suspending all its current ongoing activities (intentions) and printing in standard output a message. If a *closed* event is perceived, the agent terminates.

VI. USING JACA IN REAL-WORLD APPLICATION CONTEXTS

We are currently applying the JaCa platform in different application domains, to stress the benefits but also the weaknesses of its programming model and more in general of the proposed agent-oriented programming approach.

One of these domains is the development of distributed applications based on Service-Oriented Architecture (SOA) and Web Services (WS) in particular. In that context, agents and multi-agent systems are deserving increasing attention both from the applicative viewpoint, as an effective technique to build complex SOA applications dynamically composing and orchestrating services [33], and from the foundational viewpoint, as a reference meta-model for the service-based approach, as suggested by the W3C Web Services Architecture reference document [34]. To this end, programming models and platforms are needed to build SOA/WS applications as agent-oriented systems in a systematic way, exploiting the existing agent languages and platforms to their best, while enabling their co-existence and fruitful co-operation. In that context, we devised a library of artifacts on top of the JaCa

platform, enabling the development of SOA/WS applications in terms of workspaces populated by agents and artifacts. Agents encapsulate the responsibility of the execution and control of the business activities characterizing the SOA-specific scenario, while artifacts encapsulate the business resources and tools needed by agents to operate in the application domain. In particular, artifacts in this case are exploited to model and engineer those parts in the agent world that encapsulate Web Services aspects and functionalities – e.g., interaction with existing Web Services (agents as service consumers), implementation of Web Services (agents as service providers) – eventually wrapping existing non-agent-oriented code. First results of this work are available in [35].

We are also investigating the adoption of our approach for the engineering of advanced Ambient Intelligence (AmI) applications. For the AmI context, a relevant research issue concerns how to concretely program non-intrusive applications exhibiting features such as context-awareness, personalisation, adaptivity and anticipation of users' desires [36]. To this end we applied our approach for realising a typical AmI application [37]: the management of a rooms allocation problem in the context of a smart co-working space – e.g., a school, an office building, etc. – where people can book and use rooms according to their needs and to the current occupancy schedule. The application has to set an autonomous and adaptive room management behavior in accordance with: (i) the events that are currently held – e.g., regulating the room temperature in accordance with the number of the event's participants, automatically turning off the lights for teaching events involving a projector, etc. – and (ii) also on the base of rooms (re)allocation in accordance with incoming user requests—i.e., aiming at optimising the number of events the system can host at any given time. Agents as usual encapsulate the control and decision-making part of the application, in this case related to monitoring and controlling facilities in rooms as well as deciding appropriate strategies to use for dynamic rooms allocation. The artifact-based distributed environment instead has been exploited to model and interface with the physical devices in the rooms (lights, temperature controllers, etc.), to model and represent high-level shared data structures with related operations (such as registers keeping track of room participants and schedules), besides typical coordination artifacts.

Another project where our agent-oriented programming approach has been applied concerns the engineering of an agent-based Machine-To-Machine (M2M) management infrastructure. M2M refers to technologies allowing the construction of automated and advanced services and applications (e.g., smart metering, traffic redirection, and parking management) that largely make use of smart devices (sensor and actuators of different kinds, possibly connected through a Wireless Sensor and Actor Network (WSAN)) communicating without human interventions. In [38] is discussed the realisation of an agent-based infrastructure to enable the deployment of city-scale M2M applications that share a common set of devices and network services. In such infrastructure each WSAN area

```

1 !init.
2
3 +!init
4   <- focus("NotificationManager");
5     focus("SMSService");
6     focus("ViewerArtifact").
7
8 +sms_received(Source, Message) : not (state("running"))
9   <- showNotification(Source, Message,
10      "jaca.android.sms.ViewerArtifact", Id).
11
12 +sms_received(Source, Message) : state("running")
13   <- append(Source, Message).

```

Fig. 16. Source code of the Jason agent that manages the SMS notifications.



Fig. 17. The two different kinds of SMS notifications: (a) notification performed using the standard Android status bar, and (b) notification performed using the ViewerArtifact.

group is modelled through a CartAgO workspace, where an agent acting as a *gateway* collects data sent from all the agents managing and controlling M2M devices (sensors, actuators) through artifacts. Finally a dynamic pool of agents regulate the functioning of each M2M infrastructural node in accordance with the current workloads experimented in the M2M infrastructure. The governance infrastructure is evaluated using a Smart Parking Management scenario, where an M2M system monitors the parking occupation in order to reduce traffic and to guide drivers through the streets.

The final domain we are considering in this paper is the engineering of smart mobile applications, in particular for pervasive and context-aware computing scenarios. To this end, JaCa has been ported on the Android platform [39], enabling the development of Android applications using agent-oriented programming [40] [41]. The project is called JaCa-Android. Actually, besides porting the technology, JaCa-Android includes a library of artifacts that allows agents running into an Android application to seamlessly access and exploit all the features provided by the smartphone and by the Android SDK. Just to have a taste of the approach, Fig. 16 shows a snippet of an agent playing the role of smart user assistant,

with the task of managing the notifications related to the reception of SMS messages: as soon as an SMS is received, a notification must be shown to the user. A `SMSService` artifact is used to manage SMS messages, in particular this artifact generates an observable event `sms_received` each time a new SMS is received. A `ViewerArtifact` is used to show SMS messages on the screen and to keep track – by means of the `state` observable property – of the current status of the viewer, that is if it is currently visualized by the user on the smartphone screen or not. Finally, a `NotificationManager` artifact is used to show messages on the Android status bar, providing a `showNotification` operation to this end. Depending on what the user is actually doing and visualizing, the agent shows the notification in different ways. The behavior of the agent, once completed the initialization phase (lines 1-6), is governed by two reactive plans. The first one (lines 8-10) is applicable when a new message arrives and the `ViewerArtifact` is not currently visualized on the smartphone's screen. In this case, the agent performs a `showNotification` action to notify the user of the arrival of a new message using the status bar (Fig. 17, (a)). The second plan instead (lines 12-13) is applicable when the `ViewerArtifact` is currently displayed on screen and therefore the agent could notify the SMS arrival by simply appending the SMS to the received message list showed by the viewer (Fig. 17, (b)): this is done by executing the `append` operation provided by `ViewerArtifact`.

For a developer able to program using the `JaCa` programming model, moving from one application context to another is a quite straightforward experience. Indeed, she can continue to design and program the business logic of the applications by suitably defining the `Jason` agents' behavior, and she only need to acquire the ability to work with the artifacts that are specific of the new application context.

VII. TOWARDS A NEW GENERATION OF AGENT-ORIENTED PROGRAMMING LANGUAGES FOR COMPUTER PROGRAMMING

By exploiting existing agent technologies (`Jason` and `CARTAgO` in particular), `JaCa` makes it possible to concretely experiment agent-oriented programming as a general-purpose paradigm for computer programming and software development, getting in practice some of the benefits of agent-orientation described in Section III. However, the approach lacks of some fundamental features when compared to current languages for software development – such as the object-oriented ones – due to the fact that the agent programming models / technologies on which `JaCa` is based have been designed having Distributed Artificial Intelligence problems in mind, not software development in general. These missing features concern desiderata that are not crucial from an AI point of view, but from a software engineering and programming perspective.

A first important desideratum concerns *error checking*, i.e., the possibility to detect errors in programs before executing them. Not only syntax errors, but also errors concerning the

semantics of the program: examples are allocating tasks to agents that have not plans to handle them, or executing actions that are not part of the interface of an artifact, or rather having agent plans that react to events that are never generated by the artifacts used in the plans. To this purpose, current AOP languages offer very limited and ad-hoc capabilities. In programming languages and software engineering, this issue is addressed by introducing a sound notion of *type* and *type systems* [42]. So designing agent-oriented programming languages with strong typing would allow for type checking programs at compile time, strongly impacting on the process of program development.

Typing is important also for the program organization and as a conceptual tool for building more clean and elegant systems. In fact, the definition of a notion of *subtyping* is the base for introducing *conceptual specialization* in program organization, and then defining a *substitutability principle* [43] also in agent-oriented programs, getting finally a safe way to extend and reuse program specifications.

Inheritance instead [43] – along more recent mechanisms such as *traits* [44] – are important features in OOP to achieve code reusability and a way to define hierarchies and compositions that relate implemented parts of a systems (such as classes). So we believe that suitable mechanisms that foster code reuse are important also for the agent-oriented paradigm, both on the agent side – for instance, making it possible to define new agents from existing ones, inheriting their capabilities (such as their plans) – and on the environment side – for instance, defining the artifact classes by extending existing ones, so inheriting their operations and observable properties.

Besides typing and inheritance, a stronger support for *modularity* [45]. Finding suitable abstractions and mechanisms to improve the modularization of agent behavior is a main issue also in current research in agent programming languages (examples are [46], [47], [48], [49], [50], [51]). In the case of `Jason` for instance, the source code inside an agent to achieve some goal is fragmented into a flat sequence of typically small plans, that trigger each other. A notion of module – similar to the notion of *capability* [47] adopted by the `JACK` platform [17] – has been recently proposed to improve `Jason` agent modularity [51]. Actually, among all the possible solutions that can be adopted for achieving a good modularity, we are interested in those that allow for contextually introducing also mechanisms for reuse such as inheritance and for keeping a strong separation between specification of the behaviors (through typing) and their (hidden) implementation.

Finally, we argue that a modern programming language designed for software development in general must necessarily have a good or seamless integration with object-oriented and functional programming, that are very strong and mature paradigms for defining and working with data structures and related purely transformational algorithms. This is not the case for existing agent programming languages, that are typically based on logic programming and do not provide a seamless and efficient support to manipulate objects.

These motivations lead us to explore the definition and development of new agent-oriented programming languages, integrating and embedding in a sound way all these features from their foundation. This is the objective of the simpAL project, whose first results are reported in [52]. simpAL is an agent-oriented programming language designed from scratch so as to embed some main ideas and features of JaCa, but taking object-oriented programming and in particular Java-like languages as reference for defining and manipulating data structures, and to integrate features like an explicit notion of typing and inheritance. Actually simpAL is not meant to replace JaCa, which we consider the reference platform—along with JaCaMo [53], which extends JaCa to support also organization-oriented programming—for exploring the development of multi-agent systems for tackling problems in Distributed Artificial Intelligence contexts.

VIII. CONCLUSION

In this paper, we discussed agent-oriented programming as an evolution of Object-Oriented Programming representing the essential nature of decentralized systems where tasks are in charge of autonomous computational entities, which interact and cooperate within a shared environment. In the state-of-the-art, agents and multi-agent systems have been explored so far mainly as an approach for tackling AI and DAI problems: in this paper we solicited a further perspective, which aims at exploring the value of agent-orientation as a programming paradigm, providing an effective level of abstraction to tackle the complexities which characterize modern programming (e.g. concurrency). In order to show in practice some of the main concepts underlying the approach, we exploited the JaCa platform, which is based on existing agent-oriented technologies—the Jason language to program agents and CArtaGo framework to program the environment. JaCa technology can be used to concretely experiment the approach for developing real-world applications tackling some of the main aspects that characterize today software system complexity, such as concurrency, distribution, reactivity, flexibility and autonomy. Finally, we shed a light on some fundamental features that are missing today in existing agent programming technologies and languages (such as typing and inheritance), which can be considered a must-have for us to investigate agent-orientation as a general-purpose paradigm for computer programming and software development. Future work will be devoted in particular to both verify the effectiveness of the approach in practice, using agent-oriented programming to tackle relevant programming problems and projects, and to improve our current models and technologies—JaCa and simpAL, in particular.

REFERENCES

- [1] Nicholas R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.
- [2] Mike Wooldridge. *An Introduction to Multi-Agent Systems*. John Wiley & Sons, Ltd, 2002.
- [3] Stuart Russell and Peter Norvig. *Artificial Intelligence, A Modern Approach*. Prentice Hall, 2009.
- [4] Herb Sutter and James Larus. Software and the concurrency revolution. *ACM Queue: Tomorrow's Computing Today*, 3(7):54–62, September 2005.
- [5] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.
- [6] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni. *Special Issue: Multi-Agent Programming*, volume 23 (2). Springer Verlag, 2011.
- [7] Rafael H. Bordini, Mehdi Dastani, and Amal El Fallah Seghrouchni, editors. *Multi-Agent Programming Languages, Platforms and Applications - Volume 1*, volume 15. Springer, 2005.
- [8] Rafael H. Bordini, Mehdi Dastani, Amal El Fallah Seghrouchni, and Jürgen Dix, editors. *Multi-Agent Programming Languages, Platforms and Applications - Volume 2*. Springer, 2009.
- [9] Alessandro Ricci and Andrea Santi. Agent-oriented computing: Agents as a paradigm for computer programming and software development. In *Proc. of the 3rd Int. Conf. on Future Computational Technologies and Applications (Future Computing '11)*, pages 42–51, Rome, Italy, 2011. IARIA.
- [10] Rafael Bordini, Jomi Hübner, and Mike Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, Ltd, 2007.
- [11] Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23:158–192, 2011.
- [12] A. S. Rao and M. P. Georgeff. BDI Agents: From Theory to Practice. In *1st Int. Conf. on Multi Agent Systems (ICMAS'95)*, 1995.
- [13] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.
- [14] The Foundation of Intelligent Physical Agents organization (FIPA) – <http://www.fipa.org>, last retrieved: July 5th 2011.
- [15] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17 (3), December 2008.
- [16] Alessandro Ricci, Mirko Viroli, and Giulio Piancastelli. simpA: An agent-oriented approach for programming concurrent applications on top of java. *Science of Computer Programming*, 76(1):37 – 62, 2011.
- [17] Nick Howden, Ralph Rönquist, Andrew Hodgson, and Andrew Lucas. JACK intelligent agents™ — summary of an agent infrastructure. In *Proc. of 2nd Int. Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, 2001.
- [18] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A BDI reasoning engine. In Rafael Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming*. Kluwer, 2005.
- [19] Henry Lieberman. The continuing quest for abstraction. In *ECOOP 2006*, volume 4067/2006, pages 192–197. Springer, 2006.
- [20] Michael David Travers. *Programming with Agents: New metaphors for thinking about computation*. Massachusetts Institute of Technology, 1996.
- [21] Bonnie Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996.
- [22] David Kirsh. Distributed cognition, coordination and environment design. In *Proceedings of the European conference on Cognitive Science*, pages 1–11, 1999.
- [23] Alessandro Ricci, Andrea Omicini, and Enrico Denti. Activity Theory as a framework for MAS coordination. In Paolo Petta, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in the Agents World III*, volume 2577 of *LNCIS*, pages 96–110. Springer, April 2003.
- [24] Mitchel Resnick. *Turtles, Termites and Traffic Jams. Explorations in Massively Parallel Microworlds*. MIT Press, 1994.
- [25] Gul Agha. *Actors: a model of concurrent computation in distributed systems*. MIT Press, Cambridge, MA, USA, 1986.
- [26] Philipp Haller and Martin Odersky. Scala actors: Unifying thread-based and event-based programming. *Theoretical Computer Science*, 2008.
- [27] Joe Armstrong. Erlang. *Commun. ACM*, 53:68–75, September 2010.
- [28] CArtaGo project web site – <http://cartago.sourceforge.net>, last retrieved: July 5th 2011.
- [29] David Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.
- [30] Y. Labrou, T. Finin, and Yun Peng. Agent communication languages: the current landscape. *Intelligent Systems and their Applications, IEEE*, 14(2):45 –52, mar/apr 1999.

- [31] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Cristiano Castelfranchi, and Luca Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, volume 1, pages 286–293, New York, USA, 19–23 July 2004. ACM.
- [32] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.*, 29:1104–1113, December 1980.
- [33] Michael N. Huhns, Munindar P. Singh, and Mark et al. Burstein. Research directions for service-oriented multiagent systems. *IEEE Internet Computing*, 9(6):69–70, November 2005.
- [34] W3C Web Service Architecture – <http://www.w3.org/TR/ws-arch/>, last retrieved: June 21th 2012.
- [35] Alessandro Ricci, Enrico Denti, and Michele Piunti. A platform for developing soa/ws applications as open and heterogeneous multi-agent systems. *Multiagent Grid Syst.*, 6:105–132, April 2010.
- [36] P. Remagnino and G. L. Foresti. Ambient intelligence: A new multidisciplinary paradigm. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 35(1):1–6, 2005.
- [37] A. Sorici, O. Boissier, G. Picard, and A. Santi. Exploiting the jacamo framework for realising an adaptive room governance application. In *Proc. of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE'11, AOOPEs'11, NEAT'11, & VMIL'11*, pages 239–242. ACM, 2011.
- [38] C. Persson, G. Picard, F. Ramparany, and O. Boissier. A jacamo-based governance of machine-to-machine systems. In *Proc. of the 10th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 12)*, Advances in Soft Computing Series. Springer, 2012.
- [39] Android Platform web site – <http://www.android.com/>, last retrieved: June 21th 2012.
- [40] Andrea Santi, Marco Guidi, and Alessandro Ricci. JaCa-Android: An agent-based platform for building smart mobile applications. In M. et al. Dastani, editor, *Languages, Methodologies, and Development Tools for Multi-Agent Systems*, volume 6822 of *LNAI*, pages 95–119. Springer, 2011.
- [41] JaCa-Android project web site – <http://jaca-android.sourceforge.net/>, last retrieved: June 21th 2012.
- [42] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Comput. Surv.*, 17:471–523, December 1985.
- [43] Peter Wegner and Stanley B. Zdonik. Inheritance as an incremental modification mechanism or what like is and isn't like. In *Proceedings of the European Conference on Object-Oriented Programming, ECOOP '88*, pages 55–77, London, UK, UK, 1988. Springer-Verlag.
- [44] Stéphane Ducasse, Oscar Nierstrasz, Nathanael Schärli, Roel Wuyts, and Andrew P. Black. Traits: A mechanism for fine-grained reuse. *ACM Trans. Program. Lang. Syst.*, 28:331–388, March 2006.
- [45] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15:1053–1058, December 1972.
- [46] M. Birna van Riemsdijk, Mehdi Dastani, John-Jules Ch. Meyer, and Frank S. de Boer. Goal-oriented modularity in agent programming. In *Proc. of the 5th Int. Joint Conf. on Autonomous agents and Multiagent systems (AAMAS'06)*, pages 1271–1278, New York, NY, USA, 2006. ACM.
- [47] P. Busetta, N. Howden, R. Ronquist, and A. Hodgson. Structuring BDI agents in functional clusters. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI*, volume 1757 of *LNAI*, pages 277–289. Springer, 2000.
- [48] L. Braubach, A. Pokahr, and W. Lamersdorf. Extending the capability concept for flexible BDI agent modularization. In *Programming Multi-Agent Systems*, volume 3862 of *LNAI*, pages 139–155. Springer, 2005.
- [49] Peter Novák and Jürgen Dix. Modular BDI architecture. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1009–1015, New York, NY, USA, 2006. ACM.
- [50] Koen Hindriks. Modules as policy-based intentions: Modular agent programming in GOAL. In *Programming Multi-Agent Systems*, volume 5357 of *LNCS*, pages 156–171. Springer, 2008.
- [51] Neil Madden and Brian Logan. Modularity and compositionality in Jason. In *Proceedings of International Workshop Programming Multi-Agent Systems (ProMAS 2009)*. 2009.
- [52] Alessandro Ricci and Andrea Santi. Designing a general-purpose programming language based on agent-oriented abstractions: the simpal project. In *Proc. of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE'11, AOOPEs'11, NEAT'11, VMIL'11, SPLASH '11 Workshops*, pages 159–170, New York, NY, USA, 2011. ACM.
- [53] Olivier Boissier, Rafael H. Bordini, Jomi F. Hbner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, (0):–, 2011.

Design by Contract for Web Services: Architecture, Guidelines, and Mappings

Bernhard Hollunder, Matthias Herrmann, Andreas Hülzenbecher

Department of Computer Science

Furtwangen University of Applied Sciences

Robert-Gerwig-Platz 1, D-78120 Furtwangen, Germany

Email: hollunder@hs-furtwangen.de, matthias.herrmann@hs-furtwangen.de, huelzena@hs-furtwangen.de

Abstract—Software components should be equipped with well-defined interfaces. With *design by contract*, there is a well-known principle for specifying preconditions and postconditions for methods as well as invariants for classes. Although design by contract has been recognized as a powerful vehicle for improving software quality, modern programming languages such as Java and C# did not support it from the beginning. In the meanwhile, several language extensions have been proposed such as Contracts for Java, Java Modeling Language, as well as Code Contracts for .NET. In this paper, we present an approach that brings design by contract to Web services. We not only elaborate a generic solution architecture, but also define its components and investigate the foundations such as important guidelines for applying design by contract. Technically, the contract expressions imposed on a Web service implementation will be extracted and mapped into a contract policy, which will be included into the service's WSDL interface. Our solution also covers the generation of contract-aware proxy objects to enforce the contract policy on client side. We demonstrate how our architecture can be applied to .NET/WCF services and JAX Web services.

Keywords—Design by contract; Web services; WS-Policy; Contract policies; WCF; JAX; Contracts for Java; Contract-aware proxies.

I. INTRODUCTION

Two decades ago, Bertrand Meyer [2] introduced the *design by contract* (DbC) principle for the programming language Eiffel. It allows the definition of expressions specifying preconditions and postconditions for methods as well as invariants for classes. These expressions impose constraints on the states of the software system (e.g., class instances, parameter and return values) which must be fulfilled during execution time.

Although the quality of software components can be increased by applying design by contract, widely used programming languages such as Java and C# did not support contracts from the beginning. Recently, several language extensions have been proposed such as *Code Contracts* for .NET [3], *Contracts for Java* [4] as well as *Java Modeling Language* [5] targeting the Java language. Common characteristics of these technologies are *i)* specific

This is a revisited and substantially augmented version of “Deriving Interface Contracts for Distributed Services”, which appeared in the Proceedings of the Third International Conferences on Advanced Service Computing (Service Computation 2011) [1].

language constructs for encoding contracts, and *ii)* extended runtime environments for enforcing the specified contracts. Approaches such as Code Contracts also provide support for static code analysis and documentation generation.

In this work, we will show how Web services can profit from the just mentioned language extensions. The solution presented tackles the following problem: *Contracts contained in the implementation of a Web service are currently completely ignored* when deriving its interface expressed in the Web Services Description Language (WSDL) [6]. As a consequence, constraints such as preconditions are not visible for a Web service consumer.

Key features of our solution for bringing contracts to Web services are:

- simplicity
- automation
- interoperability
- client side support
- feasibility
- usage of standard technologies
- guidelines.

Simplicity expresses the fact that our solution is transparent for the Web service developer—no special activities must be performed by her/him. Due to a high degree of *automation*, the DbC assertions (i.e., preconditions, postconditions, and invariants) specified in the Web service implementation are automatically translated into semantically equivalent contract expressions at WSDL interface level.

As these expressions will be represented in a programming language independent format, our approach supports *interoperability* between different Web services frameworks. For example, Code Contracts contained in a Windows Communication Foundation (WCF) [7] service implementation will be translated into a WSDL contract policy, which can be mapped to expressions of the Contracts for Java technology deployed on service consumer side. This *client side support* is achieved by generating contract-aware proxy objects. The *feasibility* of the approach has been demonstrated by proof of concept implementation including tool support.

In order to represent contract expressions in a Web service's WSDL, we will employ *standard technologies*: *i)* WS-Policy [8] as the most prominent and widely supported policy language for Web services, *ii)* WS-PolicyAttachment

[9] for embedding a contract policy into a WSDL description, and *iii*) the Object Constraint Language (OCL) [10] as a standard of the Object Management Group (OMG) for representing constraints in a programming language independent manner.

Design by contract is a useful instrument to improve service quality by imposing constraints, which must be fulfilled during execution time. As these constraints are typically expressed in a Turing complete language, arbitrary business logic could be encoded. In this paper, we also present guidelines on how to properly apply design by contract by identifying functionality, which should not be part of DbC assertions.

Before we explain our solution in the following sections, we observe that several multi-purpose as well as domain-specific constraint languages have already been proposed for Web services (see, e.g., [11], [12], [13]). However, these papers have their own specialty and do not address important features of our approach:

- Contract expressions are automatically extracted from the service implementation and mapped to an equivalent contract policy.
- Our approach does not require an additional runtime environment. Instead, it is the responsibility of the underlying contract technology to enforce the specified contracts.
- Usage of well-known specifications and widely supported technologies. Only the notions “contract assertion” and “contract policy” have been coined in this work.

The paper is structured as follows. Next we will introduce the basics of design by contract. In Section III, we will recall the problem description followed by the elaboration of the solution architecture and an implementation strategy on abstract level. Guidelines for applying design by contract will be given in the Sections V and VI. So-called contract policies will be defined in Section VII. Then we will apply our strategy to Code Contracts for WCF services (Section VIII) and Contracts for Java for JAX-WS Web Services [14] (Section IX) followed by an example (Section X). Limitations of the approach will be discussed in Section XI. The paper will conclude with related work, a summary and directions for future work.

II. DESIGN BY CONTRACT

Design by contract introduces the so-called assertions to formalize selected aspects of a software system. Assertions impose restrictions, which must be met at certain points during program execution. An assertion can either be a precondition and postcondition of a method or a class invariant. Typically, assertions constrain values of parameters and variables such as range restrictions and null values. If an assertion is violated during runtime (i.e., it is evaluated to false), this is considered to be a software bug [15].

A. Preconditions and Postconditions

Preconditions and postconditions are a means to sharpen the specification of a method. While the method’s signature determines the required parameter types, preconditions and postconditions impose further restrictions on parameter values. Formally, a precondition (resp. postcondition) of a method is a boolean expression that must be true at the moment that the method starts (resp. ends) its execution. In general, such expressions can be quite complex comprising logical (e.g. and), arithmetic (e.g. +) and relational operators (e.g. >) as well as function calls (e.g. size()).

Preconditions ensure that methods are really invoked according to their specifications. Hence, a violation of a precondition can be viewed as a software bug in the invoking client code. In contrast, a method implementation can be considered incorrect, if its postcondition is violated. This is due to the fact that the implementation does not conform to its specification.

B. Class Invariants

A class invariant is a constraint that should be true for any instance of the class during its complete lifetime. In particular, an invariant guarantees that only those instances of a class are exchanged between method invoker and its implementation that conform to the invariant constraints. Analogously to preconditions and postconditions, a class invariant is a boolean expression, which is evaluated during program execution. If an invariant fails, an invalid program state is detected.

III. PROBLEM DESCRIPTION

We start with considering a simple Web service that returns the square root for a given number. We apply Code Contracts [3] and Contracts for Java [4], respectively, to formulate the precondition that the input parameter value must be non-negative.

The following code fragment shows a realization as a WCF service. According to the Code Contracts programming model, the static method `Requires` of the `Contract` class is used to specify a precondition while a postcondition is indicated by the method `Ensures`.

```
using System.ServiceModel;
using System.Diagnostics.Contracts;

[ServiceContract]
public interface IService {
    [OperationContract]
    double squareRoot(double d);
}

public class IServiceImpl : IService {
    public double squareRoot(double d) {
        Contract.Requires(d >= 0);
        return Math.Sqrt(d);
    }
}
```

Listing 1. WCF service with Code Contracts.

The next code fragment shows an implementation of the square root service in a Java environment. In this example, we use Contracts for Java. In contrast to Code Contracts, Contract for Java uses annotations to impose constraints on the parameter values: `@requires` indicates a precondition and `@ensures` a postcondition.

```
import javax.jws.WebMethod;
import javax.jws.WebService;
import com.google.java.contract.Requires;

@WebService()
public class Calculator {
    @WebMethod
    @Requires("d >= 0")
    public double squareRoot(double d) {
        return Math.sqrt(d);
    }
}
```

Listing 2. Java based Web service with Contracts for Java.

Though the preconditions are part of the Web service definition, they will not be part of the service's WSDL interface. This is due to the fact that during the deployment of the service *its preconditions, postconditions, and invariants are completely ignored* and hence are not considered when generating the WSDL. This is not only true for a WCF environment as already pointed out in [16], but also for Java Web services environments such as Glassfish/Metro [17] and Axis2 [18].

As contracts defined in the service implementation are not part of the WSDL, they are not visible to the Web service consumer—unless the client side developer consults additional resources such as an up to date documentation of the service. But even if there would exist a valid documentation, the generated client side proxy objects will not be aware of the constraints imposed on the Web service implementation. Thus, if the contracts should already be enforced on client side, the client developer has to manually encode the constraints in the client application or the proxy objects. Obviously, this approach would limit the acceptance of applying contracts to Web services.

Our solution architecture overcomes these limitations by automating the following activities:

- Contracts are extracted from the service implementation and will be transformed into corresponding OCL expressions.
- The OCL expressions will be packaged as WS-Policy assertions—so-called contract assertions.
- A contract policy (i.e., a set of contract assertions) will be included into the service's WSDL.
- Generation of contract-aware proxy objects—proxy objects that are equipped with contract expressions derived from the contract policy.
- Usage of static analysis and runtime checking on both client and server side as provided by the underlying contract technologies.

An important requirement from a Web service development point of view is not only the automation of these activities, but also a seamless integration into widely used Integrated Development Environments (IDEs) such as Visual Studio, Eclipse, and NetBeans. For example, when deploying a Web service project no additional user interaction should be required to create and attach contract policies.

IV. SOLUTION ARCHITECTURE

In this section we introduce the components of the proposed architecture (see Figure 1). This architecture has been designed in such a way that it can be instantiated in several ways supporting both .NET/WCF as well as Java environments.

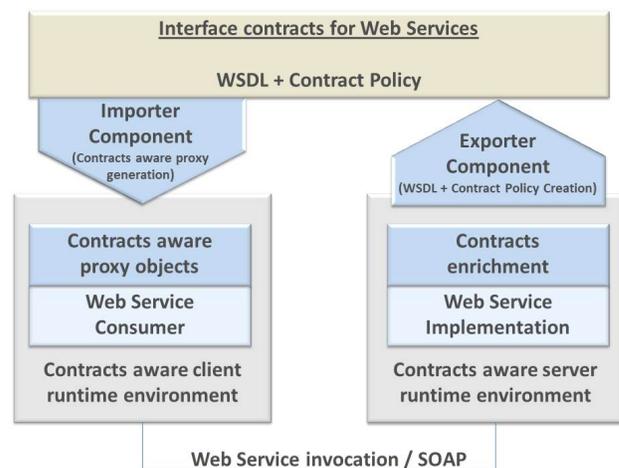


Figure 1. Solution architecture.

In short, our approach adopts the *code first strategy* for developing Web services. One starts with implementing the Web service's functionality in some programming language such as C# or Java. We assume that some contract technology is used to enhance the service under development by preconditions, postconditions, and invariants. In Figure 1, this activity is indicated by *contract enrichment*. At this point, one ends up with a contract-aware Web service such as the sample square root service at the beginning of Section III.

In order to properly evaluate the contracts during service execution, a contract-aware runtime environment is required. Such an environment is part of the employed contract technology.

We adapt the standard deployment of the Web service such that a contract policy is created and attached to the WSDL. The *exporter component* performs the following tasks:

- 1) Extraction of contract expressions by inspecting the Web service implementation.
- 2) Validation of contract expressions.

- 3) Construction of contract assertions and contract policies.
- 4) Creation of the service's WSDL and attachment of the contract policy.
- 5) Upload of the WSDL on a Web server.

Note that the generated contract policy is part of the service's WSDL and is therefore accessible for the service consumer. Both the WSDL and the contract policy is used by the *importer component* to generate and enhance the proxy objects on service consumer side. The importer component fulfills the following tasks:

- 1) Generation of the "standard" proxy objects.
- 2) Mapping of the contract assertions contained in the contract policy into equivalent expressions of the contract technology used on service consumer side.
- 3) Enhancement of the proxy objects with the contract expressions created in the previous step.

Note that service consumer and service provider may use different contract technologies. Due to the usage of OCL as "neutral" constraint language, syntactic differences between the underlying programming languages will be compensated.

V. GUIDELINES

As mentioned in Section II, the DbC principle is a useful instrument to improve service quality by imposing constraints, which must be fulfilled during execution time. As the constraints are typically expressed in a Turing complete language, DbC can be misused for specifying arbitrary business logic contradicting the idea of interface contracts. This section is concerned with the question, which functionality should (not) be realized as DbC assertions. The conformance to these guidelines can be viewed as quality checking for DbC usage.

In the following, we take a closer look to the following areas:

- 1) Side effect free operations
- 2) Validation of external input data
- 3) Exception handling with assertions
- 4) Visibility of member variables and data types
- 5) Subtyping.

A. Side Effect Free Operations

In [15], the nature of assertions is described to be *applicative*. The term emphasizes that assertions should behave like mathematical functions and hence should not produce any side effects. As a consequence, read access to resources such as member variables is feasible, however their modification is disallowed. Strictly speaking, the term *applicative* not only excludes modifications of the object the assertions applies to, but also the invocation of operations that change the state of runtime objects such as logging or console entities.

It should be noted that in DbC technologies such as *Eiffel* [19], *Code Contracts* for .NET [3], *Contracts for Java* [4]

and *Java Modeling Language* [5] assertions may invoke arbitrary functions of the underlying programming language. In their current versions, these technologies do not check the applicative nature of DbC assertions. In other words, a DbC designer does not get hints when invoking functions that directly or indirectly produce side effects.

B. Validation of External Input Data

Bertrand Meyer recommends that assertions must not be used for input validating (cf. [15]). Preconditions and postconditions are "between" the method caller and the method provider within a software component. In this sense, both caller and provider are part of the same software and do not represent an external system. Although a Web service consumer is typically part of a different software component, there is a deep logical dependency between service consumer and service provider component, which means that both components belong to the same system.

In contrast, validation of data coming from external systems should not be performed in DbC assertions. The logic for checking the quality of those data should be implemented in separate, standalone (importer) components by means of typical validation constructs such as `if/else`.

C. Exception Handling with Assertions

This aspect addresses the question how to proceed if an assertion is violated during runtime. Modern programming languages support well-known exception handling strategies based on `try/catch` blocks to locate abnormal situations and to start appropriate compensation actions.

Although current DbC technologies allow exception handling for dealing with failed assertions, a DbC designer should not intermix both techniques. Otherwise, an exception thrown by the DbC runtime environment should be handled by the surrounding application logic. This would not only have a negative impact on the overall code structure, but would also violate the first guideline "side effect free operations".

D. Visibility of Member Variables and Data Types

Web Services consumers and providers can be viewed as two parts of a common software system. However, both components are typically deployed and executed in separated runtime environments. As indicated in Figure 1, the Web service consumer sees an abstraction of the service implementation and its contained DbC assertions by means of the WSDL interface description. Thus, implementation details are not passed to the client.

For example, suppose that an assertion specified in the Web service implementation accesses a private member variable. As private member variables are not contained in a WSDL, this information is missing on consumer side. Obviously, it would not be possible to check this assertion in components, which invoke the Web service. The same

situation arises when specific data types and classes are embedded into DbC assertions, which are not available in the client environment. Thus, DbC assertions should only contain variables, data types, and methods that are meaningful and available also on Web service consumer side.

E. Subtyping

As most DbC technologies allow inheritance of preconditions and postconditions, it has to be considered how to handle such circumstances. Liskov and Wing [20] analyzed problems, which may occur in such settings. They argue that preconditions should not be strengthened by preconditions defined in derived classes. In contrast, postconditions and class invariants should not be weakened in the same way. This principle is also called *behavioral subtyping*.

Current DbC technologies apply a pragmatic approach to ensure this principle. They simply build disjunctions of the inherited preconditions. Analogously, postconditions and invariants are connected by a conjunction, which means that derived DbC assertions never become weaker (see, e.g., [21]). Of course, from a DbC developer point of view one would expect more support exceeding this basic syntactic manipulation.

VI. AUTOMATED RECOGNITION OF GUIDELINES

The automated detection of the described guidelines would be a useful feature of a DbC infrastructure. In the following, we elaborate to what extent such an approach would be feasible.

A. Side Effect Free Operations

Basically, the automated recognition of side effects is possible. We have to distinguish several cases. For example, suppose an assertion invokes a function that does not return any value (i.e., the return type is `void`). The only reason for calling this function is due to its side effects. The same is true in situations where return values are not processed in DbC assertions.

Another indicator for side effects are assignments to member variables such as `this.x = 5;`. Such statements can be easily identified by inspecting the code structure. It should be noted that some programming languages provide build-in support for declaring methods that have only limited access to resources. For example, in C++ member function can be marked with the keyword `const`, indicating that the function will not change the state of its enclosing objects [22].

In Code Contracts, the property `Pure` can be used to mark methods as side effect free. As described in [3], the current version behaves as follows: If a method not marked as `Pure` is used within an assertion, a warning is generated.

When it comes to automated recognition of side effects of entities such as logging or console, different strategies are conceivable. One strategy would be to disallow such

method calls at all, even though they are not part of the “real” business logic. Such a restrictive approach would be inline with the applicative nature of DbC assertions. One could also imagine a more liberal strategy, which allows the usage of well-defined operations (e.g., `System.out.println`, `BufferedWriter`, etc. in a Java environment). Such method calls may facilitate debugging and testing both of the software system and the attached DbC assertions.

B. Validation of External Input Data

As described in the previous section, data received from external systems should not be validated by means of DbC assertions. To check this guideline, it must be figured out, whether a specific data source should be considered external. Due to the fact that DbC technologies support function calls within assertions, data can be fetched from arbitrary sources as shown in the following listing.

```
private int userNumberInput() {
    try {
        return Integer.parseInt(new BufferedReader(new
            InputStreamReader(System.in)).readLine());
    } catch (Exception e) { return -1; }
}
@Requires({ "userNumberInput() != -1" })
public int add(int a, int b) {
    return a + b;
}
```

Listing 3. Example for validating external input.

This code fragment applies *Contracts for Java* for specifying a precondition, which depends on data read from an input stream.

In this case, it is obvious that the precondition does not have any semantic relationship to the `add` method and should therefore be avoided. However, in general there are situations, which are more complicated. For example, consider a method that processes data taken from files or network sockets. Depending on its functionality, the validation of the received data may be part of the method’s contract (and hence should be specified in a precondition) or may define some separate processing, which is only required in a specific context. Thus, an automatic compliance checker for this guideline is conceivable only in limited settings.

C. Exception Handling with Assertions

In contrast to the previous guideline, this rule can be recognized automatically. This can be achieved by analyzing the syntactic structure of the DbC assertions.

D. Visibility of Member Variables and Data Types

In general, due to the modifiers for visibility of member variables such as `private`, it could be derived automatically whether a member variable used in a DbC assertion is really meaningful to a local client.

Now suppose a client application, which invokes a Web service. As mentioned before, such a client sees the data

types and its embedded members, which are published in the WSDL of the Web service implementation. Hence, it can be checked whether a DbC assertion contains a member or a data type, which is not occurring in the WSDL. Hence, the exporter component can fully check this guideline.

E. Subtyping

Finally, the fifth guideline demands the conformance to the behavioral subtyping principle. In the previous section we observed that most DbC technologies have chosen a pragmatic approach by simply joining preconditions, postconditions, and invariants in derived classes (see, e.g., [23], [4]). However, this simple rewriting does not really solve the specification error.

In contrast, *Code Contracts* comes with a different strategy. As mentioned in [3], this technology does not allow adding any preconditions in derived types. For postconditions and invariants the behavior is similar to that of the corresponding Java technologies.

It should be noted, that a full validation of the subtyping principle is general not possible. Given two expressions (e.g., preconditions), it is in general not decidable for Turing complete DbC languages whether the one expression entails the other.

F. DbC Infrastructure Extensions

We have investigated to what extent an automated recognition of the proposed guidelines is possible. We have seen that most of the guidelines can be checked by inspecting the syntactic structure of the source code. So far, we have not implemented such a “code inspector”. The focus of our work is the elaboration of an overall solution architecture, which identifies important components for, e.g., the extraction of DbC expressions and the generation of contracts-aware proxy objects.

A conformance checker for the proposed guidelines is not part of this work. In fact, we believe that such a functionality should be provided by concrete DbC technologies. The designers of DbC implementations such as [3] and [4] can use this information to improve their approaches. Basically, we assume that a DbC compiler should produce warnings, if guidelines are violated.

VII. CONTRACT POLICIES

Having investigated important guidelines for DbC assertions, we now take a closer look to the exporter component. As shown in Figure 1, this component creates the interface contract for Web services, which is represented by a WSDL description together with a contract policy. In this section, we start with defining the building blocks of contract policies, followed by a very short introduction to the Object Constraint Language (OCL). In the final subsection, examples are given.

A. Contract Assertions

We now define contract assertions and contract policies, which allow the representation of constraints in some neutral, programming language independent format. We apply the well-known WS-Policy standard for the following reasons: WS-Policy is supported by almost all Web services frameworks and is the standard formalism for enriching WSDL interfaces. With WS-PolicyAttachment [9], the principles for including policies into WSDL descriptions are specified.

WS-Policy defines the structure of the so-called assertions and their compositions, but does not define their “content”. To represent preconditions, postconditions, and invariants, we need some language for formulating such expressions. We decided to use the Object Constraint Language (OCL) because of its high degree of standardization and support by existing OCL libraries such as the Dresden OCL Toolkit [24].

To formally represent constraints with WS-Policy, we introduce so-called *contract assertions*. The XML schema as follows:

```
<xsd:schema ...>
  <xsd:element name = "ContractAssertion"/>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name = "Precondition"
        type = "xsd:string"
        maxOccurs = "unbounded"/>
      <xsd:element name = "Postcondition"
        type = "xsd:string"
        maxOccurs = "unbounded"/>
      <xsd:element name = "Invariant"
        type = "xsd:string"
        maxOccurs = "unbounded"/>
    </xsd:sequence>
    <xsd:attribute name = "Name"
      type = "xs:string"/>
    <xsd:attribute name = "Context"
      type = "xs:anyURI"
      use = "required"/>
  </xsd:complexType>
</xsd:schema>
```

Listing 4. XML schema for contract assertions.

A *ContractAssertion* has two attributes: a mandatory *context* and an optional *name* for an identifier. The context attribute specifies the Web service to which the constraint applies. To be precise, the value of the context attribute is the name of the operation as specified in the *portType* section of the WSDL. In case of an invariant, the context attribute refers to the type defined in the *types* section.

The body of a contract assertion consists of a set of OCL expressions. Depending on the surrounding element type the expression represents a precondition, a postcondition, or an invariant. The expressions may refer to the parameter and return values of an operation as well as to the attributes of a type.

B. OCL Expressions

OCL is a formal language for specifying particular aspects of an application system in a declarative manner. Typically, OCL is used in combination with the Unified Modeling Language (UML) [25] to further constrain UML models. In OCL, “a constraint is a restriction on one or more values of (part of) an object-oriented model or system” [26]. In our context, OCL expressions will be used to specify constraints for Web services.

We use the following features of OCL in contract assertions:

- The basic types `Boolean`, `Integer`, `Real`, and `String`.
- Operations such as `and`, `or`, and `implies` for the `Boolean` type.
- Arithmetic (e.g., `+`, `*`) and relational operators (e.g., `=`, `<`) for the types `Integer` and `Real`.
- Operations such as `concat`, `size`, and `substring` for the `String` type.
- The collection types `Set` and `Sequence`.
- The construct `Tuple` to compose several values.

In order to impose restrictions on collections of objects, OCL defines operations for collection types. Well-known operations are:

- `size()`: returns the number of elements in a collection to which the method applies.
- `count(object)`: returns the number of occurrences of `object` in a collection.
- `includes(object)`: yields `true` if `object` is an element in a collection.
- `forall(expr)`: yields `true` if `expr` is true for all elements in the collection.
- `select(expr)`: returns a subcollection containing all objects for which `expr` is true.
- `reject(expr)`: returns a subcollection containing all objects for which `expr` is false.

These operations may be used to constrain admissible values for collections occurring in the service’s WSDL.

Before we give some examples, we introduce the keywords `@pre` and `result`, which can be used in postconditions. To impose restrictions on the return value of a service, the latter keyword can be used. In a postcondition, the parameters may have different values at invocation and termination, respectively, of the service. To access the original value upon completion of the operation, the parameter must be equipped with the prefix `@pre`.

C. Examples

The first example considers the square root service from Section III, extended by a postcondition. The XML fragment in Listing 5 shows a formulation as a contract assertion. The identifier `d` in the precondition refers to the parameter name of the service as specified in the WSDL.

```
<ContractAssertion context="SquareRootService">
  <Precondition>
    d >= 0
  </Precondition>
  <Postcondition>
    result >= 0
  </Postcondition>
</ContractAssertion>
```

Listing 5. Contract assertion for square root service.

The next example illustrates two features: *i*) the definition of an invariant and *ii*) the usage of a path notation to navigate to members and associated data values. Consider the type `CustomerData` with members `name`, `first name` and `address`. If `address` is represented by another complex data type with members such as `street`, `zip` and `city`, we can apply the path expression `customer.address.zip` to access the value of the `zip` attribute for a particular customer instance.

Whenever an instance of `CustomerData` is exchanged between service provider and consumer, consistency checks can be performed as shown in the following figure:

```
<ContractAssertion context="CustomerDataService">
  <Invariant>
    this.name.size() > 0
  </Invariant>
  <Invariant>
    this.age >= 0
  </Invariant>
  <Invariant>
    this.address.zip.size() >= 0
  </Invariant>
</ContractAssertion>
```

Listing 6. An invariant constraint.

To demonstrate the usage of constraints on collections we slightly extend the example. Instead of passing a single `customerData` instance, assume that the service now requires a collection of those instances. Further assume that the parameter name is `cds`. In order to state that the collection must contain at least one instance, we can apply the expression `cds->size() >= 1`. With the help of the `forall` operator one can for instance impose the constraint that the `zip` attribute must have a certain value: `cds->forall(zip = 78120)`.

VIII. CODE CONTRACTS AND WCF

We now instantiate the solution architecture presented in Section IV. We start with investigating Code Contracts for WCF (this section); the following section applies Contracts for Java to JAX Web services.

A. Exporting Contract Policies

In WCF, additional WS-Policy descriptions can be attached to a WSDL via a so-called custom binding. Such a binding uses the `PolicyExporter` mechanism also provided by WCF. To export a contract policy as described in

Section IV, a class derived from `BindingElement` must be implemented. The inherited method `ExportPolicy` contains the specific logic for creating contract policies. Details for defining custom bindings and applying the WCF exporter mechanism are described elsewhere (e.g., [16]) and hence are not elaborated here.

B. Creating Contract Assertions

Code Contracts expressions are mapped to corresponding contract assertions. Thereby we distinguish between the creation of *i*) the embedding context and *ii*) OCL expressions for preconditions, postconditions, and invariants.

In Code Contracts, a precondition (resp. postcondition) is specified by a `Contract.Requires` statement (resp. `Contract.Ensures`). Thus, for each `Requires` and `Ensures` statement contained in the Web service implementation, a corresponding element (i.e., `Precondition` or `Postcondition`) will be generated. The context attribute of the contract assertion is the Web service to which the constraint applies.

According to the Code Contracts programming model, a class invariant is realized by a method that is annotated with the attribute `Contract.InvariantMethod`. For such a method, the element `Invariant` will be created; its context is the type that contains the method.

Let us now consider the mapping from Code Contracts expressions to corresponding ones of OCL. We first observe that Code Contracts expressions may not only be composed of standard operators (such as Boolean, arithmetic and relational operators), but can also invoke pure methods, i.e., methods that are side-effect free and hence do not update any pre-existing state. While the standard operators can be mapped to OCL in a straightforward manner, user defined functions (e.g., prime number predicate) typically do not have counterparts in OCL and hence will not be translated to OCL. For a complete enumeration of available OCL functions see [10], [26].

The following table gives some examples for selected features:

Code Contracts	OCL
<code>0 <= x && x <= 10</code>	<code>0 <= x and x <= 10</code>
<code>x != null</code>	<code>not x.isType(OclVoid)</code>
<code>Contract.OldValue(param)</code>	<code>@pre param</code>
<code>Contract.Result<T>()</code>	<code>return</code>
<code>Contract.ForAll(cds, cd => cd.age >= 0)</code>	<code>cds->forAll (age >= 0)</code>

Table 1. Mapping of Code Contracts expressions to OCL.

In the first two examples `x` denotes a name of an operation parameter. They illustrate that there are minor differences regarding the concrete syntax of operators in both languages. The third example shows the construction how to access the value of a parameter at method invocation. While Code

Contracts provide a `Result` method to impose restrictions on the return value of an operation, OCL introduces the keyword `return`. In the final example, `cds` represents a collection; the expressions impose restrictions, which must be fulfilled by all instances contained in the collection.

C. Collections and Functions

OCL provides a limited set of collection types and functions. Table 2 shows how important collection types and functions of C# will be mapped to OCL.

C#	OCL
<code>System.Collections.Generic.HashSet<Of T></code>	<code>Set</code>
<code>System.Array</code>	<code>Sequence</code>
<code>System.Collections.Generic.List<Of T></code>	<code>Sequence</code>
<code>!System.Linq.Enumerable.Any()</code>	<code>Collection.isEmpty()</code>
<code>System.Collections.ICollection.Count()</code>	<code>Collection.size()</code>
<code>System.Collections.Generic.List.Add(object)</code>	<code>Sequence.append(object)</code>
<code>int % int</code>	<code>Integer.mod(int)</code>
<code>System.Math.Max(double, double)</code>	<code>Double.max(double)</code>
<code>System.String.ToUpper()</code>	<code>String.toUpperCase()</code>
<code>System.String.Concat(String)</code>	<code>String.concat(String)</code>
<code>System.String.Substring(intStart, intOffset)</code>	<code>String.substring(intStart - 1, intStart + intOffset)</code>

Table 2. Mapping types and functions from C# to OCL.

It should be noted that there are some widely used types, which are not supported by OCL. An example is a type representing dates. Such a type is part of Java (e.g., `java.util.Date`) and C# (e.g., `DateTime` in the .NET system namespace). In such a case we propose the following strategy. We define a set of “virtual” OCL types (e.g., `OCL_Date`) together with the mapping rules between the corresponding types in the programming languages such as C# and Java. Thus, we can easily extend OCL by additional types, which are typically used in DbC assertions and which should be available at WSDL interface level. Of course, the implementations of the mappings to OCL must be adapted accordingly.

D. Importing Contract Assertions

As shown in Figure 1, the role of the importer component is to construct contract-aware proxy objects. WCF comes with the tool `svcutil.exe` that takes a WSDL description and produces the classes for the proxy objects.

Note that `svcutil.exe` does not process custom policies, which means that the proxy objects do not contain contract assertions.

WCF provides a mechanism for evaluating custom policies by creating a class that implements the `IPolicyImporterExtension` interface. In our approach, we create such a class that realizes the specific logic for parsing contract assertions and for generating corresponding Code Contracts assertions. As the standard proxy class is a partial class, the created Code Contracts assertions can be simply included by creating a new file.

IX. CONTRACTS FOR JAVA FOR JAX WEB SERVICES

In this section, we consider a contract technology for Java. The principles of this description can be carried over to other Java based contracts technologies.

A. Exporting Contract Policies

In Contracts for Java [4], the preconditions, postconditions, and invariants are expressed with the annotations `Requires`, `Ensures`, and `Invariant`, respectively. An example has been given in Section III.

The reflection API of Java SE allows the inspection of meta-data. In order to access the annotations of methods we apply these API functions. Given a method (which can be obtained by applying `getMethods()` on a class or an interface), one can invoke the method `getAnnotations()` to get its annotations. Such an annotation object represents the contract expression to be transformed into an OCL expression.

Before we consider in more detail this transformation, we discuss how to create and embed contract policies into WSDL descriptions. A Web services framework provides API functions for these tasks; these functions are not standardized, though. As a consequence, we need to apply specific mechanisms provided by the underlying Web services frameworks.

Basically, the developer has to create a WS-Policy with the assigned assertions. To include the policy file into the service's WSDL, one can use the annotation `@Policy`, which takes the name of the WS-Policy file and embeds it into the WSDL. Other frameworks create an "empty" default policy, which can be afterwards replaced by the full policy file. During deployment, the updated policy will be embedded into the service's WSDL.

B. Creating Contracts Assertions

In Contracts for Java, the expressions contained in the `@requires`, `@ensures`, and `@invariant` annotations are either simple conditions (e.g., `d >= 0`) or complex terms with operators such as `&&` and `||`. As in Code Contracts, the expressions may refer to parameter values and may contain side-effect free methods with return type

`boolean`. Similar to the mapping of Code Contracts expressions, these methods will not be mapped to contract assertions (see Section VIII-B).

The following table gives some hints how to map expressions from Contracts for Java to OCL.

Contracts for Java	OCL
<code>0 <= x && x <= 10</code>	<code>0 <= x and x <= 10</code>
<code>x != null</code>	<code>not x.isType(OclVoid)</code>
<code>old(param)</code>	<code>@pre param</code>
<code>result</code>	<code>return</code>

Table 3. Mapping of selected Contracts for Java expressions to OCL.

Note that Contracts for Java currently does not provide special support for collections (such as a `ForAll` operator). Thus, a special predicate needs to be defined by the contract developer.

C. Collections and Functions

As for C#, we will also give mappings from Java collection types and functions to OCL (see Table 4).

Java type	OCL type
<code>java.util.Set</code>	<code>Set</code>
<code>java.util.Array</code>	<code>Sequence</code>
<code>java.util.List</code>	<code>Sequence</code>
<code>java.util.Collection.isEmpty()</code>	<code>Collection.isEmpty()</code>
<code>java.util.Collection.size()</code>	<code>Collection.size()</code>
<code>java.util.List.add(object)</code>	<code>Sequence.append(object)</code>
<code>int % int</code>	<code>Integer.mod(int)</code>
<code>java.lang.Math.max(double, double)</code>	<code>Double.max(double)</code>
<code>java.lang.String.toUpperCase()</code>	<code>String.toUpperCase()</code>
<code>java.lang.String.concat(String)</code>	<code>String.concat(String)</code>
<code>java.lang.String.substring(intStart, intEnd)</code>	<code>String.substring(intStart - 1, intEnd)</code>

Table 4. Data type mappings from Java to OCL.

D. Importing Contract Assertions

To obtain the (standard) proxy objects, tools such as `WSDL2Java` are provided by Java Web services frameworks. Given a WSDL file, such a tool generates Java classes for the proxy objects. In order to bring the contract constraints to the proxy class, we apply the following strategy:

- 1) Import of the contract policy contained in the WSDL description.
- 2) Enhancement of the proxy classes by Contracts for Java expressions obtained from the contract policy.

There is no standardized API to perform these tasks. However, Java based Web services infrastructures provide their specific mechanisms. A well-known approach for accessing the assertions contained in a WS-Policy is the usage of specific importer functionality. To achieve this, one can implement and register a customized policy importer, which in our case generates @requires, @ensures, and @invariant annotations for the contract assertions contained in the WS-Policy.

The second step interleaves the generated expressions with the standard proxy classes. A minimal invasive approach is as follows: Instead of directly enhancing the methods in the proxy class, we create a new interface, which contains the required Contracts for Java expressions. The proxy objects must only slightly be extended by adding an “implements” relationship to the interface created. This extension can be easily achieved during a simple post-processing activity after WSDL2Java has been called.

X. EXAMPLE

As an example, consider a weather data Web service as provided by the National Weather Service (NWS, <http://www.weather.gov>). We take a closer look to the NDFDgenByDay service, which is described as follows: *“Returns National Weather Service digital weather forecast data. Supports latitudes and longitudes for the continental United States, Hawaii, Guam, and Puerto Rico only. Allowable values for the input variable “format” are “24 hourly” and “12 hourly”. The input variable “startDate” is a date string representing the first day (Local) of data to be returned. The input variable “numDays” is the integer number of days for which the user wants data.”*

Many of the Web services provided by NWS require a latitude and a longitude both specifying the geographical point of interest. Depending on the service, additional parameters such as the start date and length of the forecast. If called successfully, such a service will return a string, which represents the weather forecast encoded in the Digital Weather Markup Language (DWML).

The following listing shows the service’s interface with WCF/C#.

```
using System.ServiceModel;
using System.Diagnostics.Contracts;

[ServiceContract]
public interface WeatherService {
    [OperationContract]
    string NDFDgenByDay(
        decimal latitude, decimal longitude,
        System.DateTime startDate,
        int numDays,
        string format);
}
```

Listing 7. WCF/C# interface of NDFDgenByDay.

With the help of the Code Contract technology we formalize some of the constraints expressed in the documentation. Listing 8 focusses on selected preconditions and postconditions of the service.

```
public class WeatherServiceImpl : WeatherService {
    public string NDFDgenByDay(
        decimal latitude,
        decimal longitude,
        System.DateTime startDate,
        int numDays,
        string format) {
        Contract.Requires(latitude > 120 && ...);
        Contract.Requires(longitude < -175 && ...);
        Contract.Requires(numDays > 0 && numDays < 8);
        Contract.Requires(format.Equals("12 hourly") ||
            format.Equals("24 hourly"));
        Contract.Ensures(
            Contract.Result<string>().Length > 0);

        // here follows the implementation
        // of the core functionality
        return ... ;
    }
}
```

Listing 8. Excerpt of a NDFDgenByDay implementation.

Given this description of the service, our approach automatically derives a representation of the DbC constraints. In particular, it creates a contract assertion with an OCL encoding of the constraints (see Listing 9).

```
<ContractAssertion context="NDFDgenByDay">
  <Precondition>
    latitude > 120 && ...
  </Precondition>
  <Precondition>
    longitude < -175 && ...
  </Precondition>
  <Precondition>
    numDays > 0 && numDays < 8
  </Precondition>
  <Postcondition>
    result.size() > 0
  </Postcondition>
</ContractAssertion>
```

Listing 9. Contract assertion for the weather service.

As described above, this contract assertion is packaged into a contract policy, which will be attached to the weather service’s WSDL. This interface description is used to generate the contract-aware clients for .NET/WCF (cf. Section VIII-D) and JAX (cf. Section IX-D).

XI. LIMITATIONS AND OPEN ISSUES

We have already mentioned that contract languages have a higher expressivity than OCL. They in particular allow the usage of user-defined predicates implemented, e.g., in Java or C#. As OCL is not a full-fledged programming language, not every predicate can be mapped to OCL. In other words, only a subset of the constraints will be available at interface level. At first sight, this seems to be a significant limitation. However, the role of preconditions and postconditions is

usually restricted to perform (simple) plausibility checks on parameter and return values. OCL has been designed in this direction and hence supports such kinds of functions.

Although WS-Policy [8] and WS-PolicyAttachment [9] are widely used standards, there is no common API to export and import WS-Policy descriptions. As mentioned before, Web services infrastructures have their specific mechanisms and interfaces how to attach and access policies. Thus, the solutions presented in this paper must be slightly adapted if another Web services framework should be used. For instance, the exporter and importer classes for processing contract policies must be derived from different interfaces; also the deployment of these classes must be adapted.

Finally, we observe that the exception handling must be changed, if contract policies are used. This is due to the fact that the contract runtime environment has the responsibility to check the constraints. If, e.g., a precondition is violated, an exception defined by the contract framework will be raised, that contains a description of the violation (e.g., that the value of a particular parameter is invalid). This must be respected by the client developer—at least during the test phase of the software application.

XII. RELATED WORK

There are several approaches that increase the expressivity of WSDL towards the specification of constraints. An overview of different constraint validation approaches is given in Frohofer [27] together with an evaluation of different implementation strategies ranging from in-place injection to wrapper- and compiler-based approaches. In particular, the impact on performance was investigated. In [28], Web services are enhanced by using DbC expressions to add behavioral information. In contrast to our work, the DbC assertions are not extracted from the Web service's implementation.

With WS-PolicyConstraints ([11], [29]) there is a domain independent, multi-purpose assertion language for formalizing capabilities and constraints as WS-Policy assertions. Basically, this language could be used to define code contract constraints. However, since WS-PolicyConstraints does not have an implementation, we use OCL expressions within contract assertions.

There are some results originating from the service monitoring area that are related to our approach. A communality is the usage of formal languages for specifying additional requirements for services not expressible within WSDL. Languages such as WS-Col (cf. [30]), WS-Policy4MASC (cf. [31]) and annotated BPEL (cf. [32]) are optimized for monitoring messages and support, for example, message filtering, logging and correlation. However, this is not the target of our approach. Instead, our focus lies on the usage of the standardized language OCL for the specification of constraints.

The formalization of security constraints for Web services is a hot topic since the early days of Web services. WS-Security [33] and WS-SecurityPolicy [12] are two well-known and widely used specifications for imposing constraints addressing encryption and attachment of signatures for Web services. These approaches do not compete with our approach, but can be applied in addition.

XIII. CONCLUSIONS

In this paper, we have elaborated a solution architecture that brings design by contract to Web services. Through our approach, important constraints (i.e., preconditions and postconditions for methods as well as class invariants) are no longer ignored, but will be included into the service's WSDL interface description. As a consequence, service consumers not only see the parameter types required to successfully invoke the Web service, but also restrictions imposed on the types such as range restrictions.

We would like to stress two important features of our approach. First, our solution is based on well-known and widely used standards such as WS-Policy and the Object Constraint Language. Hence, no proprietary frameworks are required to implement the solution architecture. Second, in order to show the feasibility of our approach, we have instantiated our architecture with two different DbC technologies and Web services frameworks.

As mentioned in Sections V and VI, the proposed guidelines can be viewed as quality checker. This means that only those expressions should be contained in DbC assertions, which are inline with the design by contract principle.

ACKNOWLEDGMENTS

We would like to thank Ahmed Al-Moayed, Varun Sud, and the anonymous reviewers for giving helpful comments on an earlier version. This work has been partly supported by the German Ministry of Education and Research (BMBF) under research contract 17N0709.

REFERENCES

- [1] B. Hollunder, "Deriving interface contracts for distributed services," in *The Third International Conferences on Advanced Service Computing*, 2011, p. 7.
- [2] B. Meyer, "Applying 'design by contract'," *Computer*, vol. 25, no. 10, pp. 40–51, oct 1992.
- [3] Microsoft Corporation, "Code contracts user manual," <http://research.microsoft.com/en-us/projects/contracts/-userdoc.pdf>, last access on 06/10/2012.
- [4] N. M. Le, "Contracts for java: A practical framework for contract programming," <http://code.google.com/p/cofoja/>, last access on 08/15/2011.
- [5] Java Modeling Language. <http://www.jmlspecs.org/>, last access on 06/10/2012.

- [6] Web Services Description Language (WSDL) 1.1. <http://www.w3.org/TR/wsdl/>, last access on 06/10/2012.
- [7] J. Löwy, *Programming WCF Services*. O'Reilly, 2007.
- [8] Web Services Policy 1.5 - Framework. <http://www.w3.org/TR/ws-policy/>, last access on 06/10/2012.
- [9] Web Services Policy 1.5 - Attachment. <http://www.w3.org/TR/ws-policy-attach/>, last access on 06/10/2012.
- [10] Object Constraint Language Specification, Version 2.2, <http://www.omg.org/spec/OCL/2.2>, last access on 06/10/2012.
- [11] A. H. Anderson, "Domain-independent, composable web services policy assertions," in *POLICY '06: Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 149–152.
- [12] WS-SecurityPolicy 1.3. <http://docs.oasis-open.org/ws-sx/wssecuritypolicy/v1.3>, last access on 06/10/2012.
- [13] A. Erradi, V. Tosic, and P. Maheshwari, "MASC - .NET-based middleware for adaptive composite web services," in *IEEE International Conference on Web Services (ICWS'07)*. IEEE Computer Society, 2007.
- [14] E. Hewitt, *Java SOA Cookbook*. O'Reilly, 2009.
- [15] B. Meyer, *Object-Oriented Software Construction (Book/CD-ROM) (2nd Edition)*, 2nd ed. Prentice Hall, 3 2000. [Online]. Available: <http://amazon.com/o/ASIN/0136291554/>
- [16] B. Hollunder, "Code contracts for windows communication foundation (WCF)," in *Proceedings of the Second International Conferences on Advanced Service Computing (Service Computation 2010)*. Xpert Publishing Services, 2010.
- [17] A. Goncalves, *Beginning Java EE 6 Platform with GlassFish 3*. Apress, 2009.
- [18] D. Jayasinghe and A. Afkham, *Apache Axis2 Web Services*. Packt Publishing, 2011.
- [19] B. Meyer, *Eiffel : The Language (Prentice Hall Object-Oriented Series)*, 1st ed. Prentice Hall, 10 1991. [Online]. Available: <http://amazon.com/o/ASIN/0132479257/>
- [20] B. H. Liskov and J. M. Wing, "Behavioral subtyping using invariants and constraints," Tech. Rep., 1999.
- [21] Y. Feldman, O. Barzilay, and S. Tyszberowicz, "Jose: Aspects for design by contract80-89," in *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on*, sept. 2006, pp. 80 –89.
- [22] B. Stroustrup, *C++ Programming Language, The (3rd Edition)*, 3rd ed. Addison-Wesley Professional, 6 1997.
- [23] G. T. Leavens, "Jml's rich, inherited specifications for behavioral subtypes," in *Formal Methods and Software Engineering: 8th International Conference on Formal Engineering Methods (ICFEM)*. Springer-Verlag, 2006.
- [24] Dresden OCL: OCL support for your modeling language, <http://www.dresden-ocl.org/>, last access on 06/10/2012.
- [25] Unified Modeling Language (UML), Infrastructure, <http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF> last access on 06/10/2012.
- [26] J. Warmer and A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison Wesley, 2003.
- [27] L. Frohofer, G. Glos, J. Osrael, and K. M. Goeschka, "Overview and evaluation of constraint validation approaches in Java," in *ICSE '07: Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 2007.
- [28] M. L. Reiko Heckel, "Towards contract-based testing of web services," *Electronic Notes Theoretical Computer Science*, vol. 82, pp. 145–5156, 2005.
- [29] "WS-PolicyConstraints: A domain-independent web services policy assertion language," Anderson, Anne H., 2005.
- [30] P. P. Luciano Baresi, Sam Guinea, "WS-Policy for service monitoring," in *Technologies for E-Services*. Springer, 2006.
- [31] A. Erradi, P. Maheshwari, and V. Tosic, "WS-Policy based monitoring of composite web services," in *Proceedings of European Conference on Web Services*. IEEE Computer Society, 2007.
- [32] S. G. Luciano Baresi, Carlo Ghezzi, "Smart monitors for composed services," in *Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC 2004)*, 2004, pp. 193–202.
- [33] WS-Security. <http://www.oasis-open.org/committees/wss>, last access on 06/10/2012.

An Implementation Approach for Inter-Cloud Service Combination

Jie Tao, Daniel Franz, Holger Marten, and Achim Streit
Steinbuch Center for Computing
Karlsruhe Institute of Technology, Germany
{jie.tao, holger.marten, achim.streit}@kit.edu, daniel2712@gmx.de

Abstract—Increasing cloud platforms have been developed in the recent time and are provisioning different services to customers. The existence of different cloud provides offers the users an opportunity of combining different cloud services to run their scientific workflows with lower cost and higher performance. In this work we developed a framework to allow combining individual services of different clouds to solve complex problems with efficiency. The framework contains a workflow management system that processes users workflow descriptions and starts the related services automatically on the target clouds. A data management component takes care of the data exchange between the underlying clouds. Moreover, a prediction model computes the potential cost and performance of running a workflow on existing clouds, giving users help in selecting the execution target for better performance/cost effect.

Keywords-Service Composition, Cloud Computing, Workflow Engine, Cloud Interoperability

I. INTRODUCTION

Cloud Computing is an emerging technology for providing IT capacities as services. Increasing number of customers is using the resources, such as computational power, software, storage, and network, offered by various cloud providers for their daily computation requirement. However, inter-cloud computing is still a novel topic. We developed a workflow framework to enable the interaction across different cloud infrastructures [1].

The service concept has been proposed for several decades. However, the term of Cloud Computing was known first in 2008 as Amazon published its Elastic Compute Cloud (EC2) [2] and Simple Storage Service (S3) [3]. Cloud Computing became thereafter a hot topic in both industrial and academic areas. There exist different definitions of Cloud Computing, including our earlier contribution [4]. Recently, the National Institute of Standards and Technology (NIST) provides a specific definition: Cloud Computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [5].

A specific feature of Cloud Computing is that computation and storage resources are provided as services. In this way, applications/software can be executed or maintained on the cloud without the necessity of operating an own local

infrastructure. Such a computing model significantly reduces the cost for resource and software management, which is clearly an attractive benefit for small business companies and research groups. In addition to cost-efficiency, Cloud Computing shows other advantages such as on-demand resource provision, supporting legacy codes, service reliability, easy management, and so on. Therefore, more and more cloud infrastructures are established and increasing numbers of users are joining the cloud world.

The fact that different cloud providers exist brings a chance to users: combining different cloud services to efficiently solve a complex problem. However, it also introduces difficulties for deciding to choose which cloud in case of multiple identical services. Depending on the size and requirement of tasks as well as the hardware configuration of clouds the tasks may perform better on some platforms than on others.

The goal of this work is to design and prototypically implement a management system that combines different cloud services to run a user-defined workflow with an additional functionality of helping users select the cloud providers.

A workflow is a methodology that splits the computation of a complex problem into several tasks. A well-known scenario is to run scientific experiments on the Grid [6], where an entire computation is partitioned and distributed over several computing nodes with a result of being able to process large data sets. This scenario can also occur on the cloud when scientific applications move to them. Furthermore, there are other scenarios on the cloud, where users require the workflow support. For example, users may compose the services provided by different clouds to benefit from their diverse functionality and to achieve a better performance/cost ratio.

Currently, it is possible to run a workflow on the cloud [7]. However, running workflows is still limited to a single cloud platform. The partitions (called tasks) of a workflow, however, may have different behavior on different clouds, indicating a requirement of running workflows across several cloud platforms.

To enable this execution mode we developed a workflow management system with a workflow engine, a data management component, and a mathematical model for performance and cost prediction. In difference to Grid workflow

implementations that target on a unified interface [8], a cloud workflow system has to cope with different interfaces and features of individual clouds. In order to enable the combination of single workflow tasks running on various clouds, we implemented a cloud abstraction layer to deliver common access interfaces. The developed framework was tested with several sample workflows.

The remainder of the paper is organized as following. Section II gives a short introduction of Cloud Computing, together with some related work. Section III analyzes the requirement on a cloud workflow framework and presents the designed software architecture. Section IV describes our initial prototypical implementation of the workflow framework in detail. Section V shows the experimental results with sample workflows. The paper concludes in Section VI with a short summary and several future directions.

II. BACKGROUND AND RELATED WORK

Cloud Computing introduces a new computing paradigm. This paradigm uses virtualization as its fundamental technology. A traditional computer system runs applications directly on the complete physical machine. Using virtualization, applications are executed on virtual machines (VM), with each VM typically running a single application and a different operating system.

Cloud Computing distinguishes itself from other computing paradigms, like Grid Computing, Global Computing, and Internet Computing, in the following aspects:

- Utility computing model: users obtain and employ computing platforms in computing clouds as easily as they access a traditional public utility (such as electricity, water, natural gas, or telephone network).
- On-demand service provisioning: computing clouds provide resources and services for users on demand. Users can customize and personalize their computing environments later on, for example, software installation, network configuration, as users usually own administrative privileges.
- QoS guaranteed offer: the computing environments provided by computing clouds can guarantee QoS for users, e.g., hardware performance like CPU speed, I/O bandwidth and memory size. The computing cloud renders QoS in general by processing Service Level Agreement (SLA) with users.
- Autonomous system: the computing cloud is an autonomous system and it is managed transparently to users. Hardware, software and data inside clouds can be automatically reconfigured, orchestrated and consolidated to present a single platform image, finally rendered to the users.
- Security: the host system monitors the communication to the VMs, restricting the number of successful attacks. Even if an attack is effective, the attack could

only compromise one VM, while the other applications and operating systems maintain a secure state.

- Availability: VMs can be easily migrated increasing the system's fault tolerance and availability.

Currently established cloud infrastructures mainly deliver three kinds of services: Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service. IaaS targets on an on-demand provision of the computational resources. The commercial computing cloud Amazon EC2 and its non-commercial implementation Eucalyptus [9] are well-known examples of IaaS-featured cloud platforms. SaaS allows the consumers to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface [5]. An example of SaaS is Web-based email. PaaS targets on an entire platform including the hardware and the application development environment. Google App Engine [10] and Microsoft Azure [11] are examples of PaaS-featured clouds.

The concept of resource sharing in Cloud Computing is similar to Grid Computing. Cloud Computing allows on-demand resource creation and easy access to resources, while Grid Computing developed standards and provides various utilities. Table I shows several major features of both computing paradigms. A detailed comparison between Grid and Cloud Computing can be found in [12].

One utility implemented on the Grid is the workflow management system. Production Grids, such as WLCG [13], TeraGrid [14], and EGEE [15], commonly support the execution of scientific workflows on the underlying resources. There are also various implementations of workflow engines on the Grid. Examples are ASKALON [16], Unicore [17], Kepler [18], GridAnt [19], Pegasus [20], and GridFlow [21]. An overview of these workflow systems is presented in [22].

The research work on workflow management systems on the cloud has been started. A well-known project is the Cloudbus Toolkit [23] that defines a complete architecture for creating market-oriented clouds. A workflow engine is also mentioned in the designed architecture and described in detail in [24]. The authors analyzed the requirement and changes needed to be incorporated when moving scientific workflows to clouds. They also described the visions and inherent difficulties when a workflow involves various cloud services.

In the sense of service combination, there are several efforts on automated processes [25]–[27]. For service composition on the cloud research issues have been discussed [28], [29] and practices have been conducted as well [30]. In addition, there are also activities on preparing and executing workflows [31] on such composite services. Recently, scientists proposed the idea of “federated clouds” [32], which can effectively utilize several clouds to enhance the QoS.

Existing work on cloud workflow systems is either in the research phase or an implementation work within a single

Table I
CLOUD VS. GRID: A FUNDAMENTAL COMPARISON.

	Cloud Computing	Grid Computing
Objective	Provide desired computing platform via network enabled services	Resource sharing Job execution
Infrastructure	One or few data centers heterogeneous/homogeneous resources under central control Industry and business	Geographically distributed, heterogeneous resources, no central control, VO Research and academic organization
Middleware	Proprietary, several reference implementations exist, e.g., Amazon	Well developed, maintained, and documented
Application	Suited for general applications	Special application domains like High Energy Physics
User interface	Easy to use/deploy, no complex user interface required	Difficult to use and to deploy Need new user interface, e.g., commands, APIs, SDKs, services
Business model	Commercial: pay-as-you-use	Publicly funded: use for free
Enabling technology	Virtualization, SaaS, Web 2.0, Web service, ...	HPC, Grid infrastructure, middleware
QoS	Possible	Little support
On-demand provisioning	Yes	No

cloud platform. The work presented in this paper aims at a prototypical implementation of a workflow engine that enables the execution of a workflow across different cloud platforms thereby using their individual services. Such a tool is currently still not available. Our goal is to simply provide a new functionality rather than to investigate a comprehensive solution. Therefore, we majorly focus on the design of an architecture and a prototypical implementation with basic functionalities. Our main contributions are the workflow engine that enables inter-cloud service combination and the proposal of a prediction model for estimating the execution time of tasks on different clouds.

III. ARCHITECTURE DESIGN

Grid Computing has been investigated for more than a dozen of years and established standards. Cloud Computing, in contrast, is a novel technology and has not been standardized. The specific feature of each cloud brings additional challenges to implementing a workflow engine on clouds.

A. Design Challenges

Grid workflows may be executed in several resource centers but the involved resources are contained in a single Grid infrastructure and hence can be accessed with the same interface. Cloud workflows, however, run usually on different clouds.

Figure 1 shows a sample scenario of running workflows on clouds. While some tasks may be executed on the same cloud, e.g., cloud C1, some others may run on different cloud platforms. The data are transferred from one cloud to another in order to deliver the output of one task to other tasks. Unfortunately, different clouds use also different data format. Furthermore, existing clouds have their own access interfaces. A standard, called Open Cloud Computing Interface (OCCI) [33], has been proposed and several implementations are currently available. However, this standard is

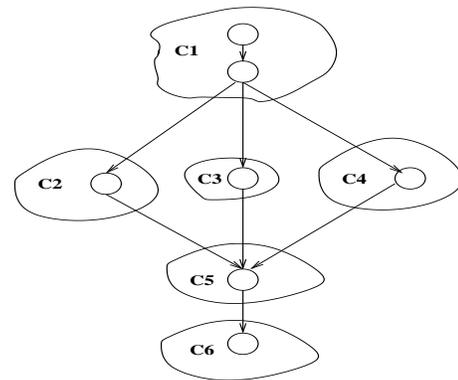


Figure 1. A sample execution scenario of cloud workflows.

only supported by a few cloud infrastructures. To link the services of different clouds, an abstraction layer is therefore required for providing an identical view with the data and interfaces of the target cloud infrastructures.

Additionally, the service price varies across cloud providers. Cloud users usually expect an optimal performance vs. cost tradeoff: i.e., acquiring the best service with the lowest payment. While increasing number of cloud infrastructures is emerging, there may be several choices to run a workflow task. A prediction model, which is capable of estimating the performance and cost of an execution on a specific cloud, can help users select an appropriate cloud for their tasks.

Based on the aforementioned observations, we designed a software architecture for the proposed cloud workflow engine and defined a performance-cost model. The following two subsections give some details.

B. Software Architecture

Figure 2 demonstrates the software architecture of the proposed workflow engine for Cloud Computing. It contains a

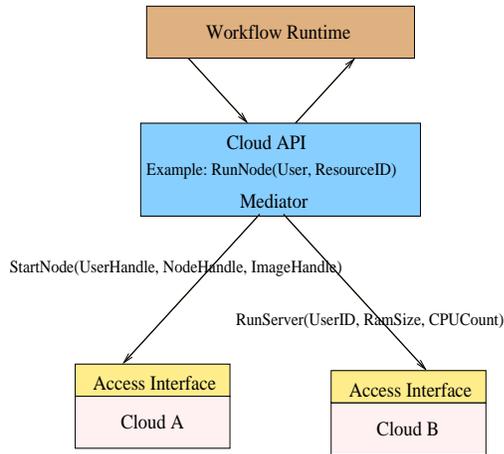


Figure 2. Software architecture of the workflow engine.

workflow runtime system, the cloud API, and the underlying cloud infrastructures. Users submit their workflows using a client side interface. Based on the description of tasks in a workflow, the execution time of the tasks on different clouds is predicted. Using this information users extend the workflow description by specifying a target cloud for each task. In the following, the workflow runtime handles the execution of tasks on the clouds and delivers the results to the users.

An important component in the architecture is the cloud abstraction layer, shown in the middle of Figure 2. The task of this layer is to implement a unified API for accessing different clouds. The runtime environment of the workflow engine uses this API to run the tasks in a workflow.

The abstraction layer defines common functions for cloud activities. It also contains a mediator that translates the functions in the unified API to concrete calls to the underlying cloud platforms. For example, the function `RunNode()` is provided for running a virtual machine instance on any IaaS-featured cloud. During the runtime the mediator replaces the function by a cloud specific one, in this example, either `StartNode` for cloud A or `RunServer` for cloud B. It also maps the function parameters in the functions of the unified API to the functions of the APIs of individual clouds. Furthermore, the mediator handles the authentication/security issues.

C. Prediction Model

Cloud users not only take care of the execution performance but pay more attention to the payment for using resources on the clouds. As an initial design, we bring the two most important metrics, application execution time and the cost, into the prediction model. Workflows in this work are defined as: A workflow is comprised of several tasks, each is combined with an application/software that is either

executed on an IaaS-cloud or hosted as a Web service on a SaaS/PaaS-cloud.

The execution time of a workflow (EoW in short) can be calculated with the following mathematical form:

$$EoW = EoT_1 + DT_1 + EoT_2 + DT_2 + \dots + EoT_n \quad (1)$$

where EoT_i is the execution time of task i and DT_i is the time for transferring data from T_i to T_{i+1} . Note that we ignore the time to start a service on the cloud as well as data transfers from and back to the customer environment.

The execution time of a single task depends on the features of the host machine on which the task is running. Roughly, it can be presented with:

$$EoT = f(S_{comp}, F_{cpu}, S_{mem}, S_{I/O}) \quad (2)$$

where the parameters are size of the computation, frequency of CPU, size of memory and cache, and size of the input/output data. For parallel applications, an additional parameter, the communication speed, has to be considered.

The price of a service on a cloud is usually determined by the node type and the location of the resource. Each cloud provider maintains a price table, where concrete payment (in US\$ per hour) is depicted. Based on this table, we calculate the cost of a workflow task with:

$$CoT = f(EoT, \$/h) \quad (3)$$

The cost of executing a workflow is then calculated as following:

$$CoW = CoT_1 + CoT_2 + \dots + CoT_n \quad (4)$$

The functions for computing the execution time of a task can be designed differently with a tradeoff between complexity and accuracy. We implemented a simple model, which is detailed in the next section.

IV. INITIAL IMPLEMENTATION

Based on the designed architecture described above, we implemented a prototype of the cloud workflow framework. This initial implementation focused on the following components:

- Cloud abstraction
- Runtime execution environment and data management
- Prediction model

A. Cloud Abstraction

To run a workflow on diverse clouds, an abstraction layer is required for the purpose of hiding the different access interface each cloud presents to the users. We use jClouds [34] as the base of this work. jClouds provides a framework for transferring programs and their data to an IaaS-cloud and then starting an instance to execute the program on the cloud. The current release of jCloud can connect several IaaS-clouds including Amazon EC2. There are other IaaS cloud

abstraction libraries, such as libcloud [35] and deltacloud [36]. We use jCloud as the base of this work due to its feature of supporting SSH and a number of cloud providers.

jClouds defines an API for accessing the underlying IaaS platforms. For SaaS/PaaS-featured clouds, however, there exists currently no implementation for an abstraction layer. Our main task in extending jClouds is to develop an S+P abstraction that interacts with SaaS-featured and PaaS-featured clouds.

The S+P abstraction contains two kinds of functions, GET and POST, for transferring data and service requests. Their input and output are defined in XML documents. This is identical to all clouds. Each cloud, however, requires specific input and output formats as well as different parameters for service requests. Our solution is to use XSL Transformation (XSLT) [37] to map the input and output of the service functions to the required data format and service parameters.

XSLT is a part of the Extensible Stylesheet Language (XSL) family and usually adopted to transform XML documents. An XSLT file describes templates for matching the source document. In the transformation process, XSLT uses XPath, an interface for accessing XML, to navigate through the source document and thereby to extract information or to combine/reorganize the separate information elements. For this work an XSLT document is introduced for some data formats, like SOAP. For others, such as binary and JSON (JavaScript Object Notation), a data transformation is not needed.

SaaS/PaaS services are started via HTTP-based protocols. A service request is sent to the service URL via GET/POST. Information like Cookies, SOAP actions and other parameters can be wrapped in the message head. The content of the call is either contained in the body of the HTTP message in case of POST calls or coded in the URL in case of GET calls.

The process of invoking a SaaS or PaaS service with the developed S+P abstraction contains the following steps:

- Processing the input data of the service request.
- Constructing a URL for the service. Information about Cookies, SOAP actions, and other parameters, is contained in the head of the message, while the body of the message defines the request.
- A service request is sent to the aforementioned URL, together with the data.
- The results of the service are downloaded as raw data. For the data formats like SOAP, where the results are coded, an XSLT document is defined to extract the useful information.

B. Workflow Execution and Inter-Cloud Data Transfer

In order to allow an easier understanding of the tasks for a cloud workflow execution engine, we take a simple workflow as an example. Figure 3 demonstrates the sample workflow consisting of eight tasks, T-a to T-f, which are combined

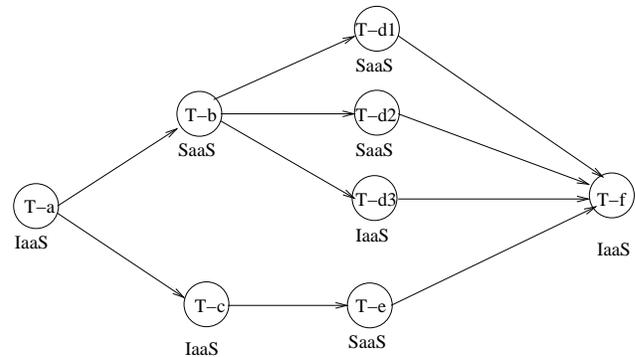


Figure 3. A simple cloud workflow.

through a respective data flow. A task can be a program or an available Web service on a SaaS or PaaS cloud. For the former, the program is executed on an IaaS cloud, while for the latter the cloud provides resources for running the software. The workflow and its tasks are defined by the user in the following XML file:

```

<workflowdefinition>
  <task name="a">
    <input filetype="some/type" name="in" />
    <output filetype="some/type" name="oa" />
  </task>
  <task name="b">
    <input filetype="some/type" name="oa" />
    <output filetype="some/type" name="ob" />
  </task>
  <task name="c">
    <input filetype="some/type" name="oa" />
    <output filetype="some/type" name="oc" />
  </task>
  <task name="d1">
    <input filetype="some/type" name="ob" />
    <output filetype="some/type" name="od1" />
  </task>
  <task name="d2">
    <input filetype="some/type" name="ob" />
    <output filetype="some/type" name="od2" />
  </task>
  <task name="d3">
    <input filetype="some/type" name="ob" />
    <output filetype="some/type" name="od3" />
  </task>
  <task name="e">
    <input filetype="some/type" name="oc" />
    <output filetype="some/type" name="oe" />
  </task>
  <task name="f">
    <input filetype="some/type" name="od1" />
    <input filetype="some/type" name="od2" />
    <input filetype="some/type" name="od3" />
    <input filetype="some/type" name="oe" />
    <output filetype="some/type" name="of" />
  </task>

```

```

<workflow from="a" to="b" />
<workflow from="a" to="c" />
<workflow from="b" to="d1" />
<workflow from="b" to="d2" />
<workflow from="b" to="d3" />
<workflow from="c" to="e" />
<workflow from="d1" to="f" />
<workflow from="d2" to="f" />
<workflow from="d3" to="f" />
<workflow from="e" to="f" />
</workflowdefinition>

```

The workflow definition mainly describes the input and output of each task and thereby also specifies the data flow between the tasks. In the concrete example the root element *workflowdefinition* is specified by the tasks to be executed, where each task is combined with an input node and an output node. The last task *f*, for example, has four input nodes describing the source of its input as shown in Figure 3. The last few lines of the XML document define the data flow between individual tasks, with help of the *workflow* element. The data flow from task a to task b and task c, from task b to task d1, task d2, and task d3, and so on.

The tasks, however, are not defined in the workflow document but described in a separate configuration file. An example is a task for extracting texts from a video. This task is used in a language translation workflow that will be introduced in the next section. The following XML file defines this task:

```

<task name="videototext" type="IaaS">
  <binary name="julius" parallel="
    sequential">
    <input type="video/flv" param="" />
    <output type="text/plain" param="" />
  </binary>
</task>

```

In the example, the IaaS task *videototext* is defined with a binary called *julius* that will be sequentially executed on an IaaS cloud. The type of its input and output is also specified in the XML file.

The primary work of the workflow execution engine is to interpret the workflow definition and starts each of the tasks on a cloud, based on the configuration XML files. In addition, the engine is also responsible for transferring the result of one task to its successor and downloading the final results to the user. The first task is performed within a single cloud and contains the following steps, which are all covered by the cloud abstraction described above:

- Transferring data (program or service parameters) to the target cloud.
- Executing the program on an IaaS cloud or invoking the Web service on the SaaS or PaaS cloud. In the case of IaaS, a virtual machine instance has to be started and some scripts are executed for configuration and program installation.

- Extracting the results out of the cloud.

To deliver the output of one task to the next task as input, the workflow execution engine has to interact with both participating clouds and to handle the inter-cloud communication. We implemented mechanisms for the following data transfers:

- IaaS to SaaS/PaaS: We use SSH to transfer data from the IaaS node to the local host and then use HTTP to deliver the data further to the SaaS/PaaS request;
- SaaS/PaaS to SaaS/PaaS: Data are extracted from the HTTP stream, stored temporarily on the host, and then applied to the next HTTP request;
- SaaS/PaaS to IaaS: Locally storing the data, which are again extracted from an HTTP stream, and then transferring them to the IaaS node via SSH;
- IaaS to IaaS: We transfer the data directly from one IaaS node to the other that is potentially located on a different cloud. This is an optimization for removing the overhead caused by an intermediate storage.

Finally, the result of the entire execution is downloaded to the user or stored on the last cloud.

C. Performance & Cost Prediction

The proposed prediction model, as described in the previous section, involves several hardware parameters that can be only acquired at the runtime by accessing the cloud resources. For the prototypical implementation, we developed a simple model without using the runtime resource information of the underlying infrastructures.

Our model is based on the execution history of similar tasks, which are defined as tasks executing the same program. The execution history is stored in a user database that contains the following main data structures:

- tasks: identifies each individual task with a unique ID and the associated program.
- I/O: defines the size of input and output files of tasks.
- node_class: describes a computing node with node ID, node name, cloud name, payment cycle, and startup time.
- execution: describes an execution of a task on a node with several attributes including program name, node_class, size of I/O, and execution time.
- node_price: gives the per-cycle-price of the computing nodes.
- node_location: gives the country and continent the node is located.

We use an SQLite [38] database system to store the data structures. Figure 4 shows a screenshot of the SQLite database browser, where the data structures and the stored data are presented. The database browser allows us to simply modify the data items and to view the data. As examples, Figure 5 and 6 show the node and execution data collected during our tests. The browser depicts the data in the form of tables.

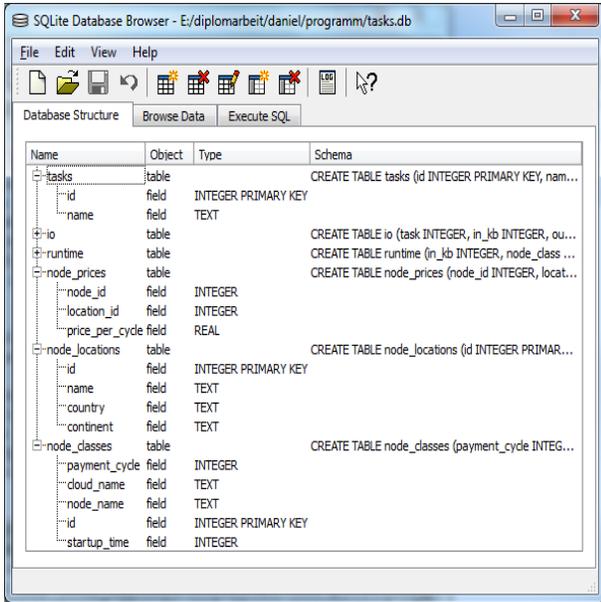


Figure 4. The data structures in an SQLite database.

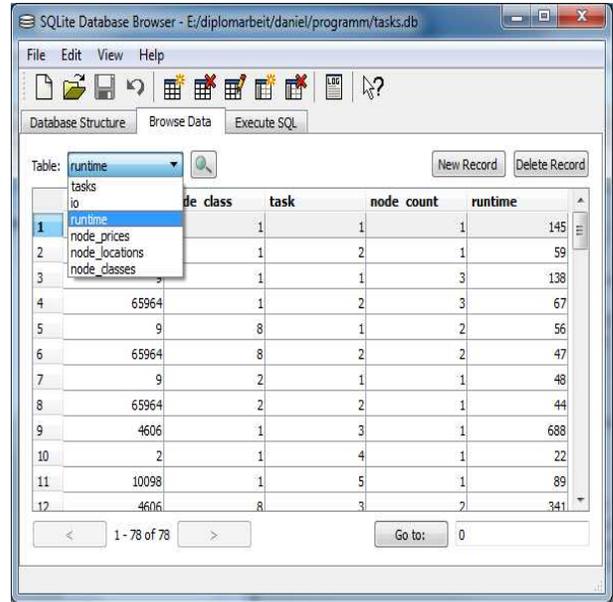


Figure 6. Task execution information stored in the SQLite database.

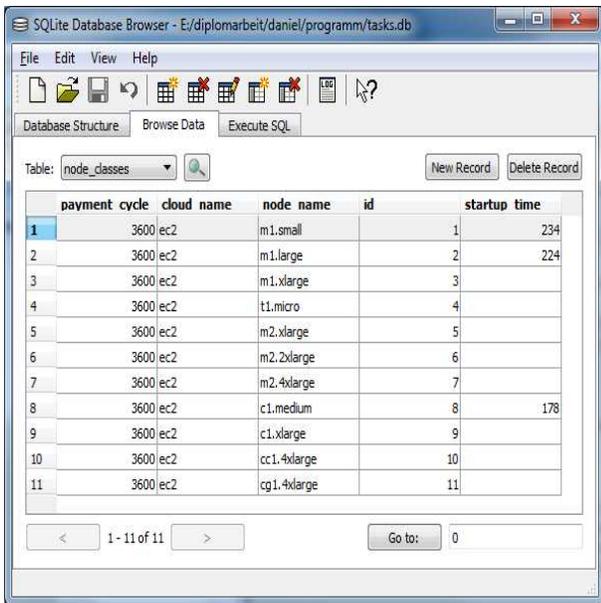


Figure 5. Node class information stored in the SQLite database.

The concrete table in Figure 5 lists all possible computing nodes on the Amazon EC2. EC2 uses a unit of one hour to calculate the cost of running instance leading to a payment cycle of 3600 (in seconds). We tested three different EC2 nodes and the startup time of these nodes can be seen in the last column of the table.

We also tested different tasks on various computing nodes. The table in Figure 6 depicts the execution information of the first twelve tests, with the data size of the task on the

first column, the ID of the target computing node on the second, the task ID on the third, the number of nodes used for the test on the fourth, and the execution time on the last column. The sixth line, for example, shows that task 2 with an input data of 65964 bytes was executed on two EC2 “c1.medium” nodes (node ID 8) in 47 seconds.

For each task in a new user-defined workflow, the potential execution time is calculated for all registered clouds and their associated computing nodes. The payment is then calculated according to the price published by the cloud providers. The first five {cloud, node} pairs with the best performance vs. cost tradeoff are shown to the users to help them select the optimal target platforms.

We use the following algorithm to predict the execution time of a new task presented with $t_{(p,s)}$, where the first attribute is the program to be executed and s is the size of the input data.

First, the average execution time of the program on a node n_s is calculated with the following equation:

$$t_{(p,n_s)} = \frac{\sum_{i=1}^n t_{i(p,n_s,s_i)}}{n}$$

where $t_{i(p,n_s,s_i)}$ is the time measured with the recorded i th execution of program p on node n_s with a data size of s_i , and n is the number of executions. Here, $t_{(p,n_s)}$ is associated with the average data size $s_{(p,n_s)}$, which is calculated in a similar way. The execution time of the new task $t_{(p,s)}$ can then be estimated with

$$t_{(p,s)} = \frac{s}{s_{(p,n_s)}} \cdot t_{(p,n_s)}$$

Table II
AN EXAMPLE OF EXECUTION HISTORY.

Tests	Size of data (KB)	Execution time (seconds)
1	300	15
2	100	12
3	200	13
4	250	14
5	300	14

We introduce a weight variable W_{data} to represent the influence degree of the input size on the execution time. The value of this variable may vary from task to task and shall be chosen based on experimental results.

Now we show an example for a better understanding of the prediction model. The sample data is depicted in Table II, which contains five history executions on a computing node. The size of data is presented in Kbytes while the execution time in seconds.

According to the data in the table, we acquire the calculation results as:

$$t_{(p,n_s)} = \frac{15 + 12 + 13 + 14 + 14}{5} = 13.6$$

$$s_{(p,n_s)} = \frac{300 + 100 + 200 + 250 + 300}{5} = 230$$

The execution time of task $t_{(p,150)}$ on the related computing node is:

$$\frac{\frac{150}{230} \cdot 13.6}{0.7} = 12.67$$

where we assume an I/O influence of weight 0.7 to the execution of the program.

V. EXPERIMENTAL RESULTS

To evaluate the developed framework, several workflows were tested. In this section, we present the results with two examples. The first workflow processes 3D scenes with a result of creating a video. The second workflow performs film synchronization whereby to translate the spoken text from Japanese to English.

The first workflow, depicted in Figure 7, contains two main tasks, *3dscenestopictures* (the raytracer) and *picturestovideo*. The raytracer acquires a scene file and a camera file as input and splits the scene into single pictures based on the position defined in the camera file. The single pictures are then processed by the second task to produce a continuous video.

We apply the Tachyon [39] raytracer for the first task which needs an MPI cluster on an IaaS cloud because the software is parallelized with MPI. To combine the pictures to a video, the program FFmpeg [40] is applied. We run this task on a single IaaS node. The workflow is defined in the following XML file:

```
<workflowdefinition>
  <task name="3dscenestopictures">
    <input filetype="*.dat" name="scene" />
    <input filetype="*.cam" name="camera" />
    <output filetype="image/ppm" name="images"
      "/>
  </task>
  <task name="picturestovideo">
    <input filetype="image/ppm" name="images"
      "/>
    <output filetype="video/avi" name="video"
      "/>
  </task>
  <workflow from="3dscenestopictures"
    to="picturestovideo" />
</workflowdefinition>
```

The second workflow, as shown in Figure 8, is comprised of four components: the language identifier (task *videototext*), a translator (task *translatejatoen*), the text synthesizer (task *texttospeech*), and the task *jointovideo*. The language identifier acquires a video file as input and outputs its text in Japanese. The output is then delivered to the language translator, where an English text is produced. In the following, the text synthesizer converts the text to speech, which is combined with the video via the last task of the workflow.

We apply the language identifier Julius [41] to process the audio that is extracted from the video by FFmpeg. In order to speed up the process, an audio is first partitioned and the partitions are then processed in parallel. Hence, an MPI cluster is required for this task. For language translation, the translation service of Google is applied. In order to model a SaaS to SaaS data transfer and to verify our cloud abstraction, the Japanese text is first translated to German and then to English. The task *texttospeech* is implemented using the speech synthesizer eSpeak [42]. Finally, the aforementioned FFmpeg program combines the audio with the video. The workflow definition is as following:

```
<workflowdefinition>
  <task name="videototext">
    <input filetype="video/flv" name="video"
      "/>
    <output filetype="text/plain" name="
      jatext" />
  </task>
  <task name="translatejatode">
    <input filetype="text/plain" name="jatext"
      "/>
    <output filetype="text/plain" name="
      detext" />
  </task>
  <task name="translatedetoen">
    <input filetype="text/plain" name="detext"
      "/>
    <output filetype="text/plain" name="
      entext" />
  </task>
  <task name="texttospeech">
    <input filetype="text/plain" name="entext"
      "/>
  </task>
```

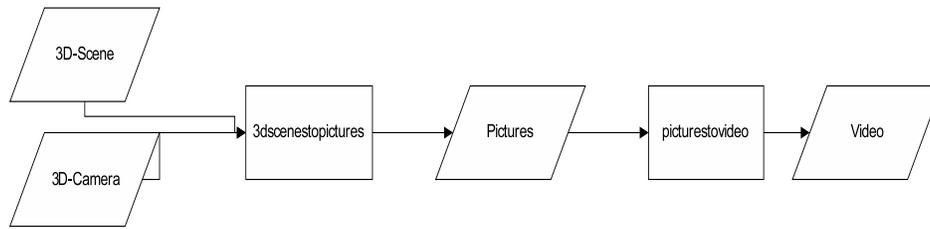


Figure 7. Data flow of the 3D-render workflow.

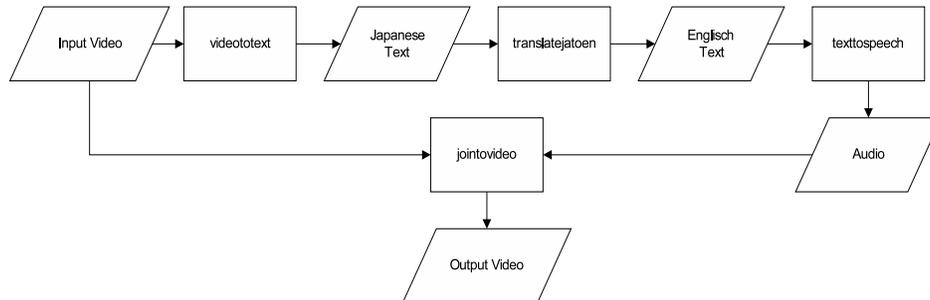


Figure 8. Data flow of the workflow film synchronization.

```

<output filetype="audio/wave" name="audio" />
</task>
<task name="jointvideo">
  <input filetype="video/flv" name="video" />
  <input filetype="audio/wave" name="audio" />
  <output filetype="video/flv" name="resultvideo" />
</task>
<workflow from="videototext" to="translatejatoen" />
<workflow from="translatejatoen" to="translatedetoen" />
<workflow from="translatedetoen" to="texttospeech" />
<workflow from="texttospeech" to="jointvideo" />
</workflowdefinition>
  
```

For the experiments we requested an account on EC2. The test results are shown in Table III and Table IV for each workflow. The tables show the execution time of tasks of a single workflow on different nodes of EC2. In the case of Google, the Web service is executed on a Google machine, which cannot be specified by the user.

The execution time of a task is presented with the measured time and the predicted one, where the former was acquired at runtime by measuring the duration of a task from submission to termination and the latter was calculated using the developed prediction model. It can be seen that the accuracy of our model varies between the tasks, where the

value with the second workflow is relative better. For the 3D render, the model underestimates the execution time in most cases, while an alternating behavior can be seen with the second workflow. Altogether, we achieved the best case with a difference of 3.4% between the real execution time and the predicted one, while the worse case shows a value of -21.2%. The difference is caused by the fact that the time for executing a program can vary significantly from one execution to the other, even though the executions are performed successively. This indicates that a more accurate model is required for a better prediction, which shall be our future work.

The values in the last column of the tables are calculated by multiplying the real execution time by the payment. It is expected that both the execution time and the payment are low. Hence, we use the values in the last column to represent the performance vs. cost tradeoff, where a lower value indicates a better behavior. Observing Table III it can be seen that the nodes "m1.small" have a better behavior. This may be associated with the concrete tasks, which do not demand a high computation capacity. With larger programs, e.g., the task *videototext* in the second workflow, a node with higher capacity, "m1.large" in this case, behaves better. However, the best choice is to use the free services provided by some clouds, such as the translation service on Google.

VI. CONCLUSIONS AND FUTURE WORK

As various cloud platforms are emerging, it is possible for users to involve several clouds to run a complex job,

Table III
EXPERIMENTAL RESULTS WITH THE 3D RENDER WORKFLOW (85 CAMERA POSITIONS).

Task	Node	Execution time			Performance vs. Cost
		Measured	Predicted	Difference (%)	
3dscenetopictures	m1.small	145	138	-4.8	12.40
	c1.medium	56	52	-7.1	19.03
	m1.large	48	42	-12.5	17.97
picturetovideo	m1.small	59	48	-18.6	5.01
	c1.medium	47	37	-21.2	15.97
	m1.large	44	36	-18.2	14.96

Table IV
EXPERIMENTAL RESULTS WITH THE WORKFLOW OF SYNCHRONIZING A FOUR MINUTES VIDEO.

Task	Node	Execution time			Performance vs. Cost
		Measured	Predicted	Difference (%)	
videototext	m1.small	665	688	3.4	168.04
	c1.medium	341	355	4.1	116.30
	m1.large	257	271	5.4	87.20
translatejatoen		45	40	-11.1	0
texttospeech	m1.small	26	22	-15.4	1.87
	c1.medium	22	20	-9.1	7.47
	m1.large	19	17	-10.5	6.46
jointovideo	m1.small	89	104	16.8	7.60
	c1.medium	87	94	8.0	29.60
	m1.large	97	75	-12.4	33.02

for example, a workflow. For this functionality, users need a framework to manage the execution of single tasks on different clouds and to deliver the result of one task to another task that may run on a different infrastructure.

We designed and implemented such a workflow management system for cloud users. The main components of the framework includes a workflow engine for task execution, a cloud abstraction to enable the interoperability of different cloud platforms, a data management component that handles inter-cloud communications, and a prediction model for estimating the cost and performance of running the workflow tasks on different cloud nodes. Initial experiments on the prototypical implementation showed the functionality of the framework.

In the next step of this work we will improve the prediction model with involvement of the runtime information of the cloud platforms. We will also provide the users with a more friendly interface to describe their workflows. A resource broker is planned as well to serve as a mediator for users to detect the best cloud services.

REFERENCES

- [1] D. Franz, J. Tao, H. Marten, and A. Streit, "A Workflow Engine for Computing Clouds," in *Proceedings of the 2nd International Conference on Cloud Computing, GRIDS, and Virtualization (CLOUD COMPUTING 2011)*. ISBN: 978-1-61208-153-3, Roma, Italy, September 2011.
- [2] "Amazon Elastic Compute Cloud," [Online], <http://aws.amazon.com/ec2/> (accessed: 2012-06-20).
- [3] "Simple Storage Service," [Online], <http://aws.amazon.com/s3/> (accessed: 2012-06-20).
- [4] L. Wang, M. Kunze, and J. Tao, "Performance evaluation of virtual machine-based Grid workflow system," *Concurrency and Computation: Practice & Experience*, vol. 20, pp. 1759–1771, October 2008.
- [5] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," [Online], http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf (accessed: 2012-06-20).
- [6] B. Asvija, K. V. Shamjith, R. Sridharan, and S. Chattopadhyay, "Provisioning the MM5 Meteorological Model as Grid Scientific Workflow," in *Proceedings of the International Conference on Intelligent Networking and Collaborative Systems*, 2010, pp. 310–314.
- [7] Y. Wei and M. B. Blake, "Service-Oriented Computing and Cloud Computing: Challenges and Opportunities," *IEEE Internet Computing*, vol. 14, no. 6, pp. 72–75, 2010.
- [8] G. Fox and D. Gannon, "Special Issue: Workflow in Grid Systems," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1009–1019, 2006.
- [9] D. Nurmi, R. Wolski, C. Grzegorzczuk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-source Cloud-computing System," in *Proceedings of Cloud Computing and Its Applications*, October 2008, available: <http://eucalyptus.cs.ucsb.edu/wiki/Presentations> (accessed: 2012-06-20).
- [10] "Google App Engine," [Online], <http://code.google.com/appengine/> (accessed: 2012-06-20).
- [11] "Windows Azure Platform," [Online], <http://www.microsoft.com/windowsazure/> (accessed: 2012-06-20).

- [12] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Proceedings of the Grid Computing Environments Workshop, 2008. GCE'08*, 2008, pp. 1–10.
- [13] WLCG, "Worldwide LHC Computing Grid," [Online], <http://lcg.web.cern.ch/lcg/> (accessed: 2012-06-20).
- [14] P. H. Beckman, "Building the TeraGrid," *Philosophical transactions - Royal Society. Mathematical, physical and engineering sciences*, vol. 363, no. 1833, pp. 1715–1728, 2005.
- [15] EGEE, "Enabling Grids for E-science," [Online], project homepage: <http://www.eu-egee.org/> (accessed: 2012-06-20).
- [16] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. S. Jr, and H. L. Truong, "ASKALON: a tool set for cluster and Grid computing," *Concurrency and Computation: Practice & Experience*, vol. 17, pp. 143–169, February 2005.
- [17] M. Riedel, D. Mallmann, and A. Streit, "Enhancing Scientific Workflows with Secure Shell Functionality in UNICORE Grids," in *Proceedings of the IEEE International Conference on e-Science and Grid Computing*. IEEE Computer Society Press, December 2005, pp. 132–139.
- [18] D. Barseghian, I. Altintas, M. B. Jones, D. Crawl, N. Potter, J. Gallagher, P. Cornillon, M. Schildhauer, E. T. Borer, E. W. Seabloom, and P. R. Hosseini, "Workflows and extensions to the Kepler scientific workflow system to support environmental sensor data access and analysis," *Ecological Informatics*, vol. 5, pp. 42–50, 2010.
- [19] G. von Laszewski, K. Amin, M. Hategan, N. J. Z. S. Hampton, and A. Rossi, "GridAnt: A Client-Controllable Grid Workflow System," in *37th Hawaii International Conference on System Science*. IEEE CS Press, January 2004.
- [20] S., D. Karastoyanova, and E. Deelman, "Bridging the Gap between Business and Scientific Workflows: Humans in the Loop of Scientific Workflows," in *IEEE International Conference on eScience*, 2010, pp. 206–213.
- [21] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, "GridFlow: Workflow Management for Grid Computing," in *Proceedings of the International Symposium on Cluster Computing and the Grid*, May 2003, pp. 198–205.
- [22] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 171–200, September 2005.
- [23] R. Buyya, S. Pandey, and C. Vecchiola, "Cloudbus Toolkit for Market-Oriented Cloud Computing," in *Proceeding of the 1st International Conference on Cloud Computing*, December 2009, pp. 978–642.
- [24] S. Pandey, D. Karunamoorthy, and R. Buyya, *Cloud Computing: Principles and Paradigms*. Wiley Press, February 2011, ch. 12, pp. 321–344.
- [25] S. McIlraith, T. C. Son, and H. Zeng, "Semantic web services," *IEEE Intelligent Systems*, vol. 16, pp. 46–53, 2001.
- [26] D. Fensel and C. Bussler, "The web service modeling framework WSMF," *Electronic Commerce Research and Applications*, vol. 1, pp. 113–117, 2002.
- [27] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint driven Web service composition in METEOR-S," in *Proceedings of the IEEE International Conference on Service Computing*, 2004, pp. 23–30.
- [28] F. Tao, L. Zhang, and Y. Hu, *Cloud Manufacturing: Development and Commerce Realization of MGrid*, in *Resource Service Management in Manufacturing Grid System*. John Wiley & Sons, Inc, 2012, ch. 15, doi: 10.1002/9781118288764.
- [29] C.-H. Hsuand and H. Jin, "Services Composition and Virtualization Technologies," *IEEE Transactions on Services Computing*, vol. 4, no. 3, pp. 181–182, 2011.
- [30] S. Wang, Q. Sun, H. Zou, and F. Yang, "Particle Swarm Optimization with Skyline Operator for Fast Cloud-based Web Service Composition," *Mobile Networks and Applications*, April 2012, online available: DOI: 10.1007/s11036-012-0373-3.
- [31] "The OASIS committee, Web Services Business Process Execution Language (WS-BPEL)," [Online], http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (accessed: 2012-06-20).
- [32] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Munoz, and G. Toffetti, "Reservoir - When One Cloud Is Not Enough," *IEEE computer*, vol. 44, no. 3, pp. 45–51, March 2011.
- [33] "Open Cloud Computing Interface," [Online], <http://occi-wg.org/> (accessed: 2012-06-20).
- [34] "jclouds," [Online], <http://www.jclouds.org/> (accessed: 2012-06-20).
- [35] "Apache Libcloud – a unified interface to the cloud," [Online], <http://libcloud.apache.org/> (accessed: 2012-06-20).
- [36] "Deltacloud – an API that abstracts the difference between clouds," [Online], <http://incubator.apache.org/deltacloud/> (accessed: 2012-06-20).
- [37] M. Kay, *XSLT 2.0 Programmer's Reference*. Wrox, 3 edition, August 2004.
- [38] "SQLite," [Online], <http://www.sqlite.org/> (accessed: 2012-06-20).
- [39] J. Stone, "An Efficient Library for Parallel Ray Tracing and Animation," In Intel Supercomputer Users Group Proceedings, Tech. Rep., 1995.
- [40] "FFmpeg," [Online], <http://www.ffmpeg.org/> (accessed: 2012-06-20).
- [41] "Open-Source Large Vocabulary CSR Engine Julius," [Online], http://julius.sourceforge.jp/en_index.php (accessed: 2012-06-20).
- [42] "eSpeak text to speech," [Online], <http://espeak.sourceforge.net/> (accessed: 2012-06-20).

User-driven Service Retrieval Platform for Converged Environments

Edgar Camilo Pedraza Alarcón	Julián Andrés Zúñiga Gallego	Luis Javier Suarez Meza	Juan Carlos Corrales
GIT	GIT	GIT	GIT
University of Cauca Popayán, Colombia epedraza@unicauca.edu.co	University of Cauca Popayán, Colombia gzja@unicauca.edu.co	University of Cauca Popayán, Colombia ljsuarez@unicauca.edu.co	University of Cauca Popayán, Colombia jcorral@unicauca.edu.co

Abstract—Today, there is an abundance of information and services in heterogeneous contexts, such as converged environments (Next Generation Networks) available for end users. However, the developments of Telecommunications and Internet converged services represent a high level of complexity for users without technical skills, since user's requests are represented by complex expressions that describe the required services. Thus, the search and selection of these services depend on the ability of the user to retrieve the most suitable ones, converting this labor in an inefficient task. With this in mind, and in order to improve the time to create convergent services, this paper proposes a novel approach that supports the automatic retrieval of services in converged environments, considering functional and non-functional properties of end-user's requests in natural language. Finally, we present the prototype that implements our proposed architecture and particularly, we describe in detail the defined tasks for natural language processing (NLP).

Index Terms—automatic service retrieval; converged environments; natural language; Telecommunications and Internet services.

I. INTRODUCTION

Service retrieval that accomplishes user requests is understood as an important stage in the composition process [1], [2], this allows the user to find and use a service based on a published description of its functionality or operational parameters [3]. Currently, services retrieval offers new challenges driven by the convergence of Web and telecommunications domains around the IP protocol, enabling the use of diverse and innovative services, regardless of the customer access network [4].

As result, operators and SMEs (Small and Medium Enterprises) that provide applications, services and content, must adapt their IT (Information Technology) infrastructure in order to develop capabilities to create and deploy new converged services with low time to market [5]. Understanding converged services, as the coordination of a range of services from different vendors, such that for end user view, it is a single service [6].

The above mentioned, both with new trends in application environments, where users are important generators of content and applications, opens up towards a new paradigm in which non-technical individuals are able to design and create their own fully customized services by integrating Web and telecommunication components, an activity that years ago was

done only by expert developers due to its complexity. This context has led to the development of various projects focused on service retrieval. However, these approaches require that users specify their request with formal expression, which becomes complex for ordinary users, even more in dynamic and varied context, as converged environment (Web + Telco) [7], [8]. In many approaches, the retrieval process is limited to Web services, leaving aside other kind of resources. On the other hand, converged environments also cover resources, such as: data, Telecom capabilities, widgets, APIs, and so on, which facilitates the compliance with the user's request, but the complexity of service retrieval for end users is still very high due to the technical knowledge required.

In order to overcome the aforementioned issues, in this paper, we propose an architecture for automatic service retrieval in converged environments, considering the services functional properties (e.g., inputs, outputs, preconditions and effects) and non-functional properties (e.g., QoS, such as: availability, response time, reputation, etc) requested by the user in natural language (NL), to reduce the complexity in the creation of converged services for end users. In the description of the architecture, we focus on describing with more detail the natural language processing (NLP) mechanisms over the other functionalities because the main objective of this approach is to support the end user, This not only ensures a correct match between the user request and the system results but also provides mechanisms to maintain a relevant quality of the input (request) of the user. Finally, we present an evaluation of each task from proposed NLP module. The preliminary evaluation shows promising results in contrast to related approaches.

Typically, NLP is useful to analyze and produce semantic representations of the user's request, providing needed information to identify functional and nonfunctional properties of services. Therefore, we propose the use of NLP techniques to improve the process of service retrieval from requests made in NL. In this regard, we reduce NLP techniques to a problem of selection of possible terms, that represent the interface or functionality of a service, or terms that are used to describe a general order of execution of services (generic control flow), clarifying that our approach does not cover a formal composition process, but displays the responses of independent retrieved services after its execution in a generic

order.

To do this, we adapt existing algorithms and develop rules for selecting terms according to certain conditions and compliance with regular expressions. The result obtained after this process, is a set of terms classified into several categories, the terms are used for obtaining services with their generic execution order. Thus, with the identification of key terms for the selection of services, supported by NLP techniques and adaptation of algorithms for matching services with those terms, it is possible to make an more accurate and automatic retrieval of services available within both Web and telecommunication domains.

The remainder of this paper is structured as follows: in the next section, we review the work related to the different topics that involves the current research. We present the problem statement in Section III. In Section IV we compare our approach with an existing one showing the advantages for the user. Then, in Section V, a high-level description of the proposed architecture is presented. To provide greater clarity an example is discussed in Section VI. In section VII we present a general evaluation of the system and finally in Section VIII we conclude the paper.

II. STATE OF THE ART

retrieval can be addressed under two main approaches: syntactic and semantic. The searching of services from a syntactic approach, considers either, interfaces matching techniques (e.g., WSDL, IDL) or keyword searching [9] that require exact matches at the syntactic level between the parameters of service descriptions. This leads to deficient results in the retrieval of service, obtaining services that do not correspond to the initial search criteria. The semantic approach allows the establishment of relationships between concepts that define the functionality of services (functional properties) and additionally, considers formal descriptions constructed by non-functional properties [10], achieving a more precise description of services and improving the quality of results (retrieved services) according to user needs [11]. From the foregoing, and given the nature of the problem, the proposed solution focuses on services retrieval based on lightweight semantics and NLP.

In turn, currently, an extension of previous approaches, is being widely accepted in the Web, where services are described by simple labels (tags) for different users, which corresponds to a classification of collaborative services through labels, without hierarchy or kinship default [12], [13]. A number of studies have shown that a set of tags added to resources, is rich and compact enough to describe and characterize, with a good degree of accuracy, the main concepts they represent [13].

The most relevant related works are described in two sections according to the main components of our proposal: NLP applications and user-driven service retrieval.

A. NLP Application

One of the most popular NLP applications is Siri. Siri is a personal assistant that resides on iPhone 4S and performs a

range of tasks understanding natural speech. Siri is based on artificial intelligence and NLP mechanisms, is able to know which application to open based on natural language requests. Working on the operating system level [14], with information from contacts, music library, calendars, reminders, and Apple's data centers, Siri is able to understand the requests and return responses [15]. Siri works with the built-in application of iPhone including weather, stocks, messaging, and others, but leaves aside the retrieval of external resources such as web services and some telecommunication capabilities, in addition, to our knowledge, Siri does not consider non-functional properties.

In [16], the IBM research team developed a supercomputer that performs analysis phases of natural language questions composed by hundreds of algorithms, some of these phases present a similar approach with the proposed ones in this paper. However, it requires a complex system composed of multi-core hardware processor, hardware capabilities that are still difficult to migrate to some end users devices such as PDAs or cell phones.

In the work developed in [17], the authors address the selection of Web services based on requests expressed in natural language, this solution is based on language restrictions, matching the structure of the request with predefined patterns for decomposition into blocks using keywords. Subsequently, the request is processed and transformed into a data flow and control model expressing the general logic of the new service. In addition, the authors propose the use of a common ontology and a NL dictionary, in order to relate textual fragments with functional parameters of such services. This work presents disadvantages when limiting requests to simple sentences.

Based on concepts graphs and conceptual distance measure, a solution is presented in [2]. Its purpose is to calculate the similarity between the user's request, represented by keywords, and services available in a repository. Within the linguistic analysis, different processes are performed: text segmentation, irrelevant word removal, elimination of derivatives (stemming) and grammatical corrections. The authors admit their proposals lack of dynamic adaptation at runtime.

The approach of [10] re-uses the converged services creation environment of project SPICE (Service Platform for Innovative Communication Environment) [18] to facilitate services retrieval with different types of semantic annotations. The author focuses his work on the development of an intelligent agent in charge of analyzing the application in NL to extract semantic information, specifically the goals, from which additional semantic information is derived, as inputs and outputs, which are used to retrieve services and report their order composition. However, retrieval's throughput and processing critically depend on the amount of services stored in the repository.

B. User-Driven Service Retrieval

Some tools and techniques have been developed to enhance resource retrieval (e.g., photos, videos, services) taking into account the information generated by the users, an example is the

Folksonomy Ontology Enrichment Tool (FLOR) [19], which performs an automatic semantic enrichment of collaborative tagging systems, from user generated tags, creating a semantic layer to describe the concepts of those tags and their relations. The tool reuses existing knowledge such as online ontologies indexed in the Watson Semantic Web Gateway and WordNet, the system also performs three phases that guarantees a lexical processing of terms, a sense expansion and a final semantic enrichment. The major drawback is that the tool presents high percentages of incorrect semantic enrichment, mainly due to the semantic expansion phase for the differences in term's definition of Wordnet and the online ontologies.

The prototype developed in [20] is implemented to improve the information retrieval in tag-based systems. Taking as input the tags provided by the end-users, the prototype adds system tags from semantic ontologies (WordNet or MultiWordNet), or tag clusters imported from the Flickr website, to annotate and retrieve videos previously imported from real tag-based system (Youtube). While a generic architecture is developed without delving into the different stages, the key ideas and concepts are useful in directing the enrichment of user-generated tags with sources of knowledge.

In [21], the work is based on the collaborative tagging of web services, where users annotate indexed Web services with keywords, in order to define structured collaborative tagging, differing between tags: input, output, and behavior of a service. In the proposed technique, the authors define a matchmaker for the Web services retrieval, which involves two stages: classification and service ranking. However, the language used as input is based on a system query, i.e., the user's query is realized in a formal language.

From the previous review, limitations are evident because most of them are based on Web services retrieval with functional preferences, leaving aside in first place, non-functional requirements that may provide more sense to services semantic descriptions, and second, not considering other kind of resources such as Telecommunication capabilities. Also, many works are not end-user centered, which makes them complex for ordinary users, moreover these works are based on formal languages, or they are limited to simple phrases made in natural language, therefore, flexible requests made in natural language are not considered. Apart from [17], the approaches lack in showing the retrieved services in a flow to express the logic of the request, useful for subsequent service composition process. Finally, the automatic retrieval of services in converged environments is a recent topic of research, where, considering all the above features has not yet been developed.

Therefore, we propose a novel end-user centered approach which facilitates the interaction of the user with the system through requests made without formal languages, considering techniques to expand the search in order to increase the accuracy of the services retrieval process and with a recommendation service system. Our approach also considers functional and non functional parameters of services and the arrangement of the retrieved services in a generic flow to

express the logic of the request.

III. PROBLEM STATEMENT

One of the key challenges of competitiveness of the existing telecommunications companies is the reduction of time in the process of creating new converged services. For this reason, converged services creation environment has been developed [22], [23], [24], which require formal requests with complex expressions by the developer, in order to compose services that are selected manually. The service manual selection shows that the process of creating converged services depends on the ability of the developer to select the most appropriate services, which is wasteful and inefficient work, since it is difficult for the human capacity to generate compositions that go hand in hand with the growing number of services available on the Web and Telecommunications domains.

In this vein, the creation time of converged services can be improved by reducing the complexity of the activities. Activities such as the selection of atomic services that are part of a composite service, need to be more flexible and agile [25], [26], [27]. To achieve this, it is necessary to define mechanisms to facilitate this process, such as making requests in a simpler and understandable language, with the intention of automating tasks in the identification of requirements and selection of services.

In this scenario, we propose an approach that allows the services retrieval in a converged environment that meets the end-users requirements. Therefore, the following problem statement is formulated:

How to reduce the complexity of services retrieval to end users in a converged environment, by ensuring compliance with their requirements?

IV. CHALLENGES AND SOLUTIONS

The proposed architecture considers problems found in currently user-driven systems for services retrieval that simplifies this process by allowing users make requests in natural language. To illustrate our proposal, let us consider an application created by Yahoo! Pipes [28], the application required by the user, retrieves food and beverage businesses for sale feeds from a website related with "food", also filters the content based on the title and the description that must contain the word "food" and retrieves two tagged images from Flickr [29] with the word "food" applying a similar filter. Finally, the application joins the retrieved information and organizes it to publish items for viewing. This process is outlined in Figure 1.

Thus, this application can be summarized to a request like: *"Get pictures of food and places selling food"*. However, for that, in this tool seven components were needed, also it requires additional configuration by the user, such as defining the URL of the feed website, manage filtering rules and define the retrieved information flow between components. It is also necessary that the user knows the different components and the functionality of each one, for example the user must know that Flickr is an image hosting website. The complexity and level

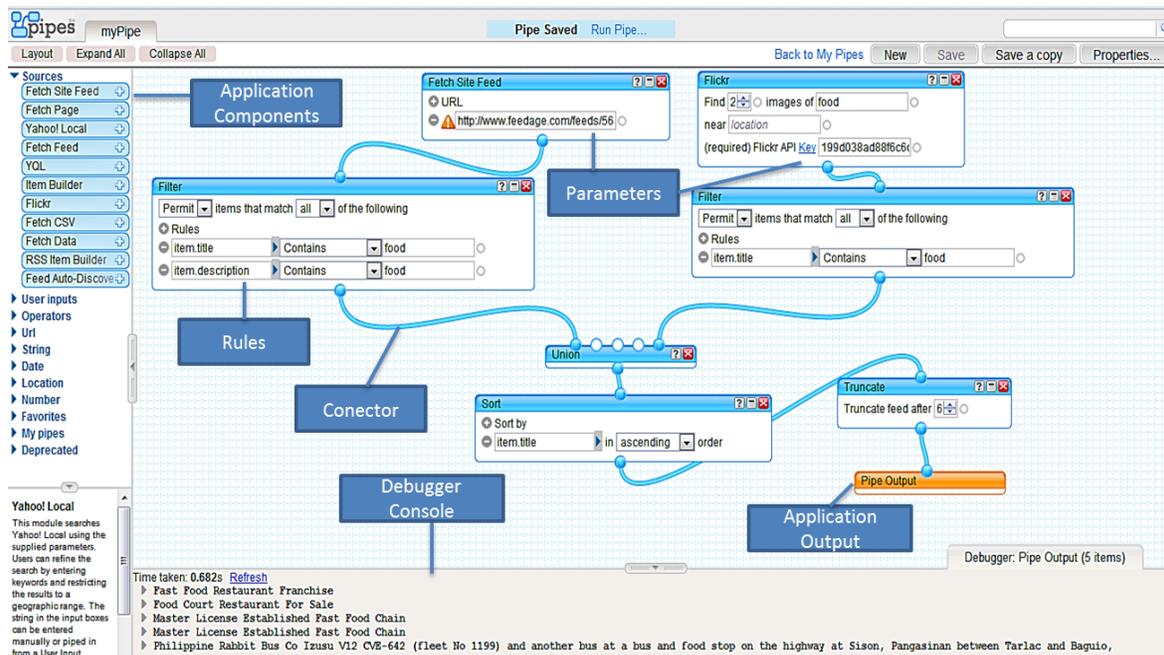


Fig. 1. Application created using YahooPipes

of knowledge required for this application is not appropriate for a common end-user.

Addition to the above restrictions, with the increase of online repositories and development of Telecommunication services, high precision service retrieval is required, also it is necessary to develop systems that are increasingly intuitive and easy to handle for users without technical skills, and that allow the automation of tasks in the identification of requirements and selection of convergent services. These challenges can be addressed considering facilities such as the use of natural language, which improves the user interaction with the system.

Also, there are other important features that should be taken into account to meet user requirements, such as speed rates of resource retrieval process or the possibility that can give a tool to allow a user to learn from others users of the tool (social-aware recommendation concept [30]). These features are considered in the system, through collaborative tagging mechanisms [21], which are an alternative solution of ontology-based approaches. Ontology-based approaches are inadequate primarily because the ontologies are expensive in creation and maintenance, also end-users are not involved in the development process of ontology, which fails to use a different vocabulary to request a service. On the other hand, collaborative tagging allows that users can see the annotation made by other users on the resources and hence extract the semantics from annotation of the community, [21]. This approach allows the handling of different kind of resources, which facilitates work in convergent environments. However, it is appropriate to clarify that due to the magnitude of the project, the description of the tagging mechanisms is omitted, and will be presented in future papers.

Our proposal allows the user to make a request as simple

as: "I want to receive pictures of food and places selling food", to retrieve components that perform this functionality. Additionally, our approach shows to the user how these components are arranged in the flow, which facilitates the interaction of the user with the system through requests made without formal languages. Likewise, our proposal recommends services during and after of the user's request and uses NLP techniques applied to the request, taking advantage of collaborative tagging and formal knowledge sources; thus, the system retrieves and publishes the outcome of the converted user's request into a components flow that represents the desired service. A Detailed explanation of the conceptual model and architecture is presented in the next section.

V. ARCHITECTURE

This section contains a detailed description of the architecture proposed, for automatic services retrieval in converged environments. The architecture is organized in four phases: Natural Language Analysis, Recommendation, Matching and Association. Phases are also composed of internal modules that carry out small process, most of them sequentially organized. The system has as input, user's request made in NL from a mobile device, and gives finally a service ranking and a generic control flow as output. In section VI, is described in more detailed the results of the system, how act this modules and how a requester can execute them. Below, architecture modules are described with its respective functionality.

- *SpellChecker*: This module receives user's requests and checks over possible mistakes, underlining misspelled words.
- *Autocomplete*: The module auto completes some words of the request, showing a possible word representing what

the user wants to write.

- *Tokenizer*: It has as input the NL request and from this, it obtains words, phrases or symbols called tokens.
- *Filter Words*: is the responsible for removing non-sense words by comparing with a set of words previously identified.
- *Words Tagging*: tags words according to its grammatical category (e.g., she "pronoun", loves "verb", animals "noun").
- *Named Entity Recognition (NER)*: It classifies the words into "functional" or "control" categories. In "functional" exists also two different categories: "input-output" and "behavior".
- *Semantic Analyzer*: in charge of semantic disambiguation process of input words, by selecting the correct meaning of a word according to request's context.
- *Normalization tags*: functions as a stemmer, obtaining their corresponding lemmas from the input words. (e.g., *doggie* or *dogs* becomes *dog*).
- *Coreferencer*: Associates possessive pronouns with the reference subject of the sentence (e.g., "*Make a call to Mary and then send a present to her*" - relates *her* with *Mary*).
- *Non Functional Requirements Recommender*: It searches non-functional parameters in the repository from functional request previously written by user.
- *Service Recommender*: It searches request-service information in the repository obtained from prior inputs of users.
- *Matcher Functional Requirements*: It obtains from cluster services, the first rank, by matching functional requirements.
- *Ranking Generator*: It obtains the final ranking of services considering, if exists, non-functional requirements, of services, such as QoS.
- *Services*: conformed by Web and Telecommunications domain services, which can be conceptually organized by their functional properties.
- *Folksonomy*: It reflects through tags the collective intelligence of a crowd or a community (*wisdom of the crowds*) in giving meaning to available resources (Power Tags) (e.g. QoS, Telco, IT, among many others), supporting functional (internal categories input-output and behavior) and non-functional properties descriptions.
- *Flow Ranking Repository*: It stores an association of service (tags) obtained at the end of the semantic matching phase.
- *Non-functional Repository*: It stores an association of non-functional and functional parameters of cluster's services.
- *Generic Flow Generator*: It generates an generic control flow, based on keywords taken from the user's request processed.
- *Flow-Ranking Associator*: It associates the flow obtained by the *Generic Flow Generator* module with the ranking output of the matching semantic phase, obtaining the final

output of the architecture.

Below we provide a more detailed description of the different stages and processes that take place in the modules of the proposed architecture (Figure 2).

A. Phase of Natural Language Analysis (NLA)

The development of the first phase can be performed through some techniques like: Gate-NLP, Open-NLP, Apache UIMA [31], [32], among others, which implement modules mentioned above. Taking into account diverse specifications and criteria such as performance, documentation and extension, we have selected the technique called GATE (General Architecture for Text Engineering [29]). This is a suite tool developed at the University of Sheffield and is a Lesser General Public License. This technique also offers an architecture that contains functionality for plugging in all kinds of NLP software.

In the same way, Gate is a platform that contains an information extraction system named A Nearly New Information Extraction System (ANNIE). This has a set of diverse modules like: sentences splitter, tokenizer, part of speech (POS) tagger, gazetteer, named entities and a co-reference tagger. Moreover, ANNIE also provides functionalities as: information extraction through Gate GUI, or can be a starting point for more specific tasks, such as machine translation, information retrieval or the one done in this project. On the other hand, Gazetteer is an important and unknown task that together with JAPE (Java Annotation Patterns Engine) are the elements used for detecting entity lists such as organizations, places, names, dates among others. In this sense, JAPE allows the establishment of grammatical rules in order to obtain words annotations, while Gazetteer contains a set of lists with words classified.

It is important to consider that GATE provides a comprehensive set of elements similar to the modules showed in this paper, however, the proposed approach adapts the modules supported by GATE and enriches them with external modules added as semantic analyzer, filtering of words, the auto-complete and spell checker module, as well as the novel approach oriented towards the use of natural language processing, to recover services which have not been developed in depth previously.

Already immerse in the architecture, we must assume initially that a user makes a request from his/her mobile device in NL. While user is typing the request, two modules are evaluating this input: in one side, the *SpellChecker* module is inspecting for misspelled words. If one word is identified as incorrect, the system underlines the word to warn user about his mistake. On the other hand, the *Auto-complete* module is constantly inspecting all the words typed, when it identifies a coincidence, between first letters of a word and a set of words stored in a repository, it will suggest a list of terms to the end user; then he can decide if he wants to auto-complete the word with the recommendations or not.

Then sentences of the request are received by the *Tokenizer* module, where tokenization operation starts, it obtains simple lexical units from complex sentences, by removing existing

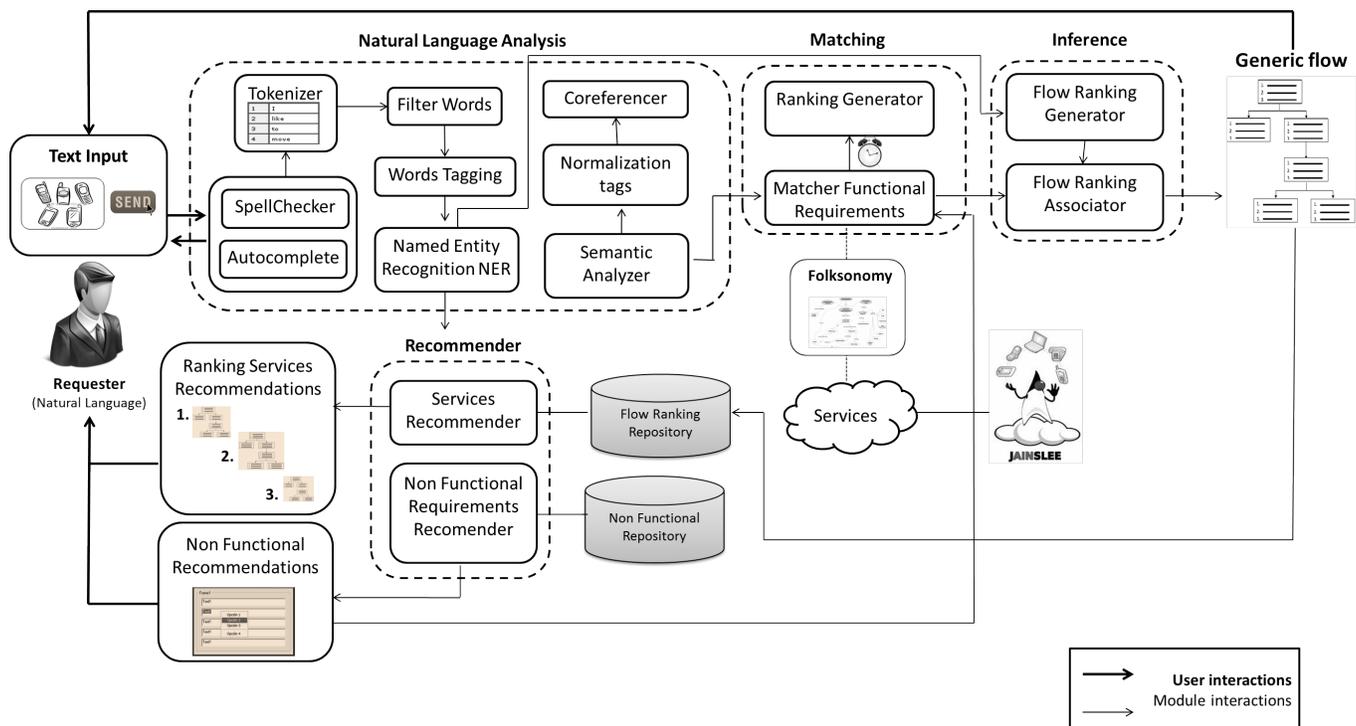


Fig. 2. Architecture of the System.

spaces or punctuation marks, admiration marks, exclamation marks and question marks. Additionally, this module corrects simple lexical errors that may arise in the request; those ignored by the user, when entered the request, or those ignored when *SpellChecker* module warned, i.e., all misspelled word that are easily identifiable. Afterward, the sentences are processed through *Filter Words* module, here, the system removes all the words that have coincidence with a set of words that have been store previously and considered as unimportant. Later, the sentence passes through *Words Tagging* module, where is pretended to classify words of the sentence according to their grammatical category. The module also aims to undertake an analysis based on linguistic rules, trying to identify and compensate syntactic and structural errors.

Once completed these operations, the request is more consistent, but remains complex. Therefore, we established the *Named Entity Recognition* module, which performs a classification between "Control" and "Functional" words according to its meaning, from which, control words are directed to *Flow Ranking Generator* module, whereas functional words are directed to *Semantic Analyzer* and *Recommender* modules. In this sense two categories have been established: input-output and behavior, both executed to identify special features of services. To perform words identification in each category, was necessary to establish some rules that allowed words category selection, e.g., numeric types are considered as input-output category. For the previous case, any number in the request would be classified and taken into account as input-output feature of service. This is achieved by name entities transducer,

where a set of rules (JAPE) are defined.

Once all the words have been selected in different categories, it is important to consider the semantic ambiguity, for which the *Semantic Analyzer* module, identifies the correct sense according to its context, i.e., it identifies the correct sense of a word with multiple meanings in a sentence. All the possible senses are extracted from Wordnet and the way we choose the correct meaning, is through Lesk algorithm [30]. Once having the correct meaning, the most important word in the definition is selected; this term will be used in the matching phase. In *Normalization tags* module, the system obtains the respective lemmas, from all the words coming from the previous module. Lemma is the canonical form of a word, and it is used as an aid in the subsequent phase of the system. In the final stage of linguistic analysis, the *co-reference* module establishes a relationship between a possessive pronoun with the main subject of the sentence. This is done in order to understand different relationships of one subject in the user's request.

All of above modules allow an easy identification of keywords, which will represent user's requests. This stage offers to users, greater flexibility in the use of language, allowing establishing a wider range of possibilities. On the other hand, the phase identifies conditional words (e.g., *if, then, later*) and order words (sequence) (e.g., *first, second*), used in Generic Flow Generator module, mentioned at association phase.

B. Recommender phase

This phase is composed of two modules and both are executed while the matching phase performs the search of ser-

vices through functional requirements. In this way, the phase initially is going to show recommendations about generic control flow with services to the user. If the user selects one recommendation, the execution of the two remaining phases would be avoided, and straightaway the process finished. Otherwise, the process continues in the matching phase. It is worth mentioning, that there is an estimated time of response, where users can select or not, any diagram recommendation. In case the time is finished, the system will understand that the user has not accepted any recommendation. The module in charged for the above task is the *Service Recommender*, which obtains a list of generic control flow with services of the repository from obtained keywords classified as functional. The second module in this phase is called *Non Functional Requirements Recommender*, this searches non-functional parameters from the *Non Functional Repository* using the obtained keywords classified as functional, the result is shown to the user, and if one or more are selected, it becomes the input of the *Ranking Generator* module, else nothing happened.

C. Matching Phase

At this stage there are two important modules, responsible for selecting the most appropriate and accurate service, according to a set of input terms: Initially, is important to consider all the described service tags, because they are semantically enriched, using different knowledge sources like Wordnet and Flickr. Thereby, is obtained a biggest set of terms that describe the services. In the same way, all input terms are semantically enriched, this process facilitates the matching process described below.

Matcher Functional Requirements use both, the terms enriched that describe services, which also are obtained from a repository, and the terms obtained from the input request. This module makes a syntactic comparison between mentioned terms, and obtains as a result an ordered list of the services that present more syntactic coincidences. In other words, this module is in charge of functional matching requirements to obtain the first service's ranking, it also considers some words classifications (input-output, behavior) and services description, that make possible the service selection process.

The second module refers to the *Ranking Generator* which use a weighting algorithm, i.e., once the first ranking of services is obtained, this module tuned the ranking with non-functional parameters, taken from the *Recommender* phase. The *Ranking Generator* module uses a timer, which provides a time out for an entry of those parameters, here user must entry those parameters which consider important to complement its request. If he/she doesn't entry anything, the system only will consider functional parameters and the ranking established would be the presented one before, otherwise, the system will obtain those non-functional parameters selected by the user.

Often users don't select the most appropriate non-functional parameters, for this reason the system will give to each non-functional parameter a weighing in the ranking generation. Although all parameters are going to be considered, the highest weighing is for those selected by end user. In summary, the

ranking of non-functional parameters is obtained taking into account the selected and non-selected parameters and their respective service values, then is generate a list of services with the best non-functional values. Finally, through a factor named tuning factor, is obtained the final ranking which joins functional and non-functional parameters. The tuning factor gives greater or lesser weight to a given ranking, depending on the configuration system input made by user. This phase is also related with a Service Logic Execution Environment SLEE, specifically JAIN-SLEE, which is characterized as an execution environment with low latency, high performance and synchronously event orientation. With this is intended that, once services are discovered and ranked, is necessary to have an execution interface to each detected service, in the case where they were going to be executed in the association phase.

We argue that requests can be enriched with non-functional parameters, in order to provide optimum results that best fit to end user requirements, since such properties are very important for better understanding of the user's request, because they represent features such as quality, efficiency, availability, etc.

D. Association phase

This is the last stage of the system and begins with the *Generic Flow Generator* module. It receives as input, terms classified as *Control words* obtained from NER through *NLA* phase. The module infers a basic structure of ordered operations that represents the basic control flow, useful for the service composition process, which is performed once the most relevant atomic services have been retrieved. The *Flow-Ranking Associator* module receives as inputs the service ranking from the semantic matching phase and the basic control flow (obtained from the previous module) in order to generate the services generic control flow (stored in *Flow Ranking Repository*) which in turn represents the output of the entire system.

VI. CASE STUDY

This section describes the functionality of the proposed architecture through an example that details each of the phases of the general process for automatic service retrieval in converged environments through natural language requests. To do so, consider the following situation. Suppose that an executive requires a service to coordinate meetings and obtain meetings reports. The executive requests from his cell phone by using natural language, the following: "I want to receive traffic reports of Bogotá via messages, minutes before the meeting and if I have not made it to the meeting, I want to receive audio content of the decisions taken."

A. Information Retrieval with Natural Language Analysis

1) *Tokenizer*: Considered as the first NLP operation to be performed in processing of the request (result 1 in the Figure 3), the result obtained by identifying each "atomic" unit is as follows. "I - Want - to - receive - traffic - reports - of - Bogotá - via - messages - , - minutes - before - the - meeting - and - if - I - have - not - made - it - to - the - meeting - , - I - want - to - receive - audio - content - of - the - decisions - taken "

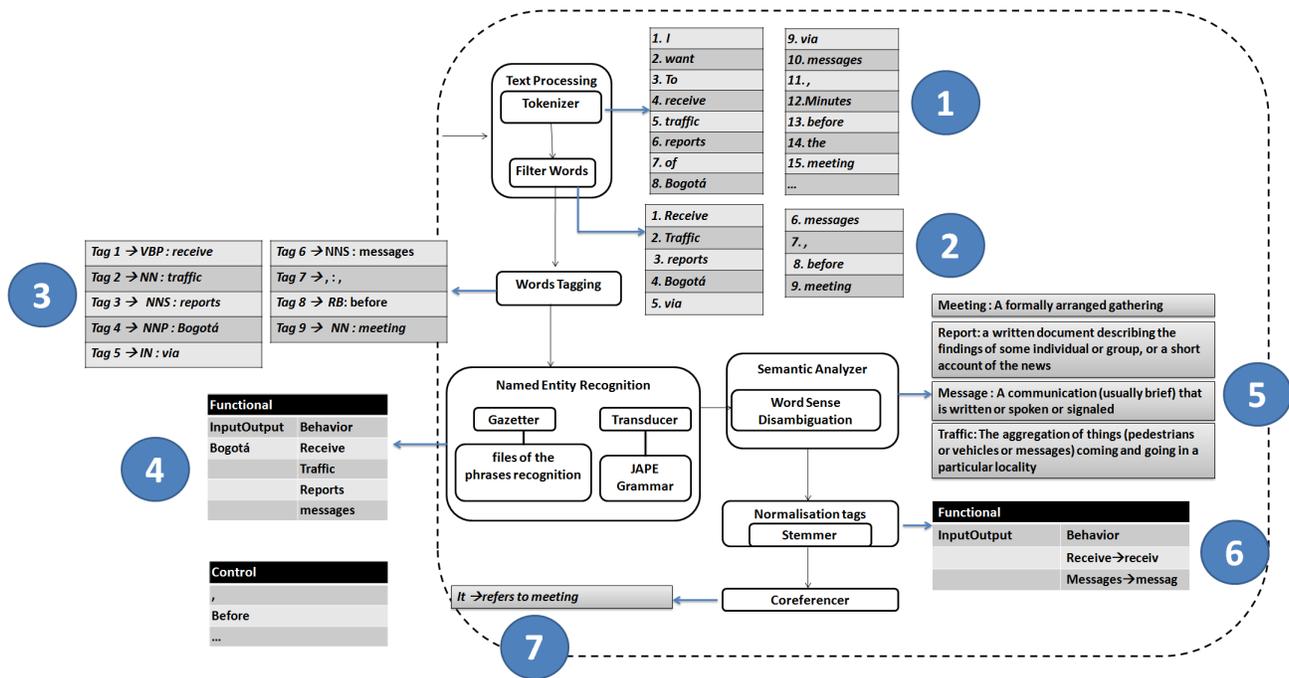


Fig. 3. Example of the Natural Language Analysis Module.

2) *Filter Words*: Following the idea of [17], it is necessary to remove common words of the request or words that do not add significant meaning, to not create noise in the final result, removing the following words: "I, want, to, of, have, to, I, want, to, of" Thus, the result is (part of the significant terms are shown as result 2 in the Figure 3): "receive - traffic - reports - Bogotá - via - messages - , - minutes - before - the - meeting - and - if - not - made - it - the - meeting - , - receive - audio - content - the - decisions - taken"

3) *Words Tagging*: Given the possibility of ambiguous words, it is necessary to perform a tagging process to help clarify this ambiguity. Thus, the words from the previous module are classified according to the grammatical category handled by GATE annotations [33]. The result obtained after performing the POST procedure (Part of Speech Tagging) is (result 3 in the Figure 3) shown in the Table I.

4) *Named Entity Recognition*: In the process to extract word entities with the annotated words resulting from the previous module, the system classifies words into functional or control categories, for the classification of functional categories, the terms are annotated based on service interface, i.e., through its inputs, outputs and behavior (service functionality). Therefore, if the term corresponds to an input or output parameter, then it is annotated as "inputoutput", and if the term corresponds to a behavior parameter, it is annotated as "behavior". The Gazetter makes the classification for matches between the terms obtained from the previous module and terms obtained from the Gazetter lists, the matching terms are annotated with *majortype* annotations, which define the type of the list, i.e., the categories "control", "inputoutput" and "behavior" and *minortype* (specific annotations), that

represent subcategories as: location, person, number, devices, food, etc. For this case in particular, the Gazetter annotates the word Bogotá with a *majortype* "inputoutput" and *minortype* "location", and annotates the terms "messages", "traffic", "reports" as part of the behavior category without additional subcategories. In addition, the gazetter finds that the word "before" is in the list called control.lst so it is also annotated with a *majortype* "control".

To complement the Gazetter, we use Java Annotation Patterns Engine (JAPE). JAPE provides finite state transduction over annotations based on regular expressions [34]. For example, we define that tokens tagged as verb by Gate (VB, VBD, VBG, VBN, and VBZ [33]) and without *majortype* annotations provided by the Gazetter, are annotated as a behavior term. The terms "receive" and "meeting", fulfill with this rule and are annotated as behavior terms. Another rule defined with JAPE is that tokens of type *punctuation* are listed as terms of control (e.g., commas are grouped in this category).

Finally, considering the order of words, blocks of terms are created in a separated way by the annotations of control, the result is shown in the Table II. (The first Block with the control terms are represented as the result 4 in the Figure 3).

5) *Semantic Analyzer*: At this point, we detail the sense disambiguation of the words classified as functional, based on dictionary definitions [35], using the Lesk algorithm [36]. This algorithm is based on the assumption that words that co-occur in a phrase tend to share the same topic. In our proposal, the disambiguation process of a term is performed with the Lesk algorithm and WordNet 1.7, which takes as input the words from the NER module and counts the number of common words between the terms from the NER module and each

TABLE I
WORDS TAGGING RESULTS

No	Tag Type	Tag
1	VBP	receive
2	NN	traffic
3	NNS	reports
4	NNP	Bogotá
5	IN	via
6	NNS	messages
7	,	,
8	RB	before
9	DT	the
10	NN	meeting
11	CC	and
12	IN	if
13	RB	not
14	VBD	made
15	PRP	it
16	DT	the
17	VBG	meeting
18	VBP	receive
19	JJ	audio
20	NN	content
21	DT	the
22	NNS	decisions
23	VCN	taken

sense of the ambiguous word given by the dictionary. Finally, the highest scoring sense (the greater number of common words) is selected. For this example, "report" can be verb or noun, and may have several meanings: *a written document describing the findings of some individual or group, a short account of the news, the act of informing by verbal, a sharp explosive sound (especially the sound of a gun firing), card a written evaluation of a student's scholarship and deportment*, etc. Thus, from which the system determines the first choice as relevant to this case. This and other disambiguated terms are shown in the result 5 of the Figure 3. Consequently, the first term of type noun, from the correct sense given by the algorithm, is added to the blocks previously created to improve the search in the matching module; therefore, in the example, the word "document", which correspond to the sense of the term "report", is added to the terms to be used in the matching and is also added to the category that belongs to the term "report", i.e., the category behavior.

TABLE II
NAMED ENTITY RECOGNITION RESULTS

Block	Category	Terms
1	Functional-Behavior	Receive
		Traffic
		Reports
		Messages
	Functional-InputOutput-Location	Bogotá
Control	,	
2	Functional-InputOutput-Time	Before
	Control	Minutes

6) *Normalization Tags*: The system uses the Porter algorithm inside GATE, provided through the snowball project [37] to make the process of stemming. The Porter algorithm removes affixes using rules and lists of words following the

pattern of algorithms known as affix removals [38]. Although the algorithm used in the system is one of the most simple, has proven to be as good as other algorithms in assessments of *precision* and *recall* for information retrieval [39].

In the example, the terms of interest to normalize are those classified as *behavior* or *inputoutput*. These terms represent the request made by the executive. The rules used by the algorithm are based on the "measure" (m) of the word, which corresponds to the number of vowels that are followed by a consonant character; for example, the term "receive" when m is greater than 1, the algorithm replaces the suffix e by null, resulting "receiv". Other terms such as "messages" and "minutes" can accomplish several conditions of the algorithm: initially the S suffix of these terms is replaced by null, resulting "message" and "minute" terms respectively, where the final result given by the algorithm are the terms "messag" and "minut". Other terms are treated similar depending on their fulfillment with the rules defined by the algorithm, some of these terms are shown as the result 6 in the Figure 3. Finally, the terms are added to the blocks as the terms of the semantic analyzer.

7) *Coreferencer*: With this module, the system generates additional annotations of type co-reference, such annotations indicate the pairs pronoun/entity, where the entity is the antecedent that refers to the pronoun. In the example, "it" in the context "if I have not made it to the meeting" has a notation (Behavior type ENTITY_MENTION_TYPE with matches in the position [118, 125]) that corresponds to the word "meeting" indicating that the pronoun "it" refers to "meeting".

B. Recommender

1) *Service Recommender*: The terms classified as functional are used to search recommendations in the *Flow Ranking Repository*, from previous results made by other users. A feature has been added to the system, common in lot of web tools and services. It consists in a simple autocomplete feature which takes as input the terms stored in the Flow Ranking Repository to recommend terms of service while the user types the request. Consider that while the user texts "want to receive traff", the system will suggest the option "traffic" according to previous requests made by other users stored in the repository. Another kind of recommendation comes after sending the request to the system, considering the case for the existence of some match with the words: "traffic reports" the result shown to the executive is a list of generic flows with services, outcome from previous requests by other users of the system (see Figure 4).

If the executive doesn't select any recommendation, so the remaining processing continues

2) *Non Functional Requirements Recommender*: This recommender searches non functional parameters from the *Non Functional Repository* using the terms classified as functional, the result considering the case for the existence of matches with the words: "traffic reports" is as follows: "Precision, real-time, cost ..." The executive chooses the option "precision" and

Recommendation 1 Recommendation 2

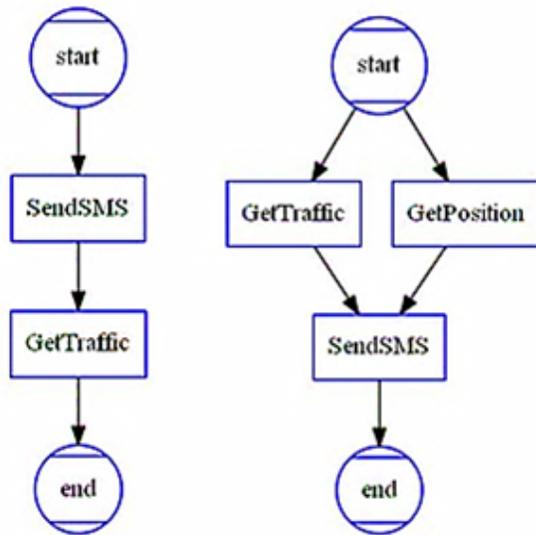


Fig. 4. Results of the Service Recommender Module

it is accepted by the system.

C. Semantic comparison between the processed request and Service Cluster

1) *Matcher functional requirements*: The input for this stage is represented by the output given by the NLA module (Table III). At this point, we consider that social software

TABLE III
OUTPUT GIVEN BY THE NLA MODULE

Block	Category	Terms
1	Functional-Behavior	Receive
		Traffic
		Reports
		Messages
		Receive
		Message
		Document
		Communication
		Aggregation
		Report
	Functional-InputOutput	Bogotá
2	Functional-InputOutput	Minutes
		Minut
		Time
3	Functional-Behavior	Meeting
		Gathering
		Meet
4	Functional-Behavior	Meeting
		Meet
		It
5	Functional-Behavior	Receive
		Audio
		Content
		Taken
		Receiv

has some disadvantages (ambiguity, variation of terms, and flat organization of tags). Works as [40] exceed the limitations with

TABLE IV
RELEVANT SERVICES FOR THE RANKING PROCESS

Block	Service Name	Description
1	PostItService	Creates messages and receive new answers.
	ReportingService	Provides report data and network usage data.
	QueueService	Offers a reliable, highly scalable hosted queue for storing messages as they travel between computers.
	DealService_v1	WebService to receive newest deals, top deals, categories, and so on.
	TopLabService	Answers on messages.
	USWeather	Get five day weather report for a given zipcode.
2	Airport	Gets Airport Code, City-OrAirport Name, Country, Country Abbv, CountryCode,GMT Offset Runway Length in Feet, Runway Elevation in Feet.
	HeaderTimeService	Returns the current time.
	StrikeIronRealTimeQuery	Retrieves information for the stocks that are traded on the Island ECN marketplace.
3	CommitteeMeetingService	Information on committee meetings of the Washington State Legislature.
4	CommitteeMeetingService	Information on committee meetings of the Washington State Legislature.
5	PostItService	Creates messages and receive new answers.
	Perceval_VoIP_manage	Downloads message audio file.
	AWSECommerceService	Exposes Amazon's product data and e-commerce functionality.

semantic technologies, so that knowledge sources enrich the tagspaces semantically. In [41], the approach addresses the fact that one tag can sometimes be mapped to different Wordnet synsets, where each synset corresponding to one meaning of the tag. Following the former idea, our system enriches the terms obtained from the *Natural Language Analyzer* module, expanding the search with sources of knowledge (Flickr and Wordnet).

In this way, the limitations of social software are partially overcome. Hence, from the terms obtained, the system classifies services: a service is classified as relevant or not depending on whether it shares some terms of input, output or behavior (service interface) with the terms of the expanded user search. Once the algorithm for classification of services is applied, a relevant set of services to each block is obtained. The services used in this example are taken from seekda information system [42], the result is presented in the Table IV.

Note that, the classification component uses the obtained services for the ranking process. Thus, the algorithm gives more importance to those services that match both input-output and behavior parameters with the user's search terms than those that only match one of the two categories. The result of

TABLE V
RANGING OF F SERVICES CONSIDERING ONLY FUNCTIONAL
PARAMETERS

Block	Service Ranking
1	1. ReportingService 2. QueueService 3. DealService_v1 4. PostItService 5. TopLabService 6. USWeather report
2	1. HeaderTimeService 2. StrikeIronRealTimeQuery 3. Airport
3	1. CommitteeMeetingService
4	1. CommitteeMeetingService
5	1. Perceval_VoIP_manage 2. PostItService 3. AWSECommerceService

TABLE VI
RANGING OF F SERVICES CONSIDERING ONLY FUNCTIONAL
PARAMETERS

Block	Service Name
1	1. ReportingService 2. DealService_v1 3. QueueService 4. PostItService 5. USWeather 6. TopLabService
2	1. StrikeIronRealTimeQuery 2. HeaderTimeService 3. Airport
3	1. CommitteeMeetingService
4	1. CommitteeMeetingService
5	1. Perceval_VoIP_manage 2. PostItService 3. AWSECommerceService

the ranking process considering only the functional parameters is shown in the Table V.

2) *Ranking Generator*: The generated ranking is subjected to a tuning based on the non-functional parameters chosen by the user. In this case, the executive chose the parameter "precision" of three available parameters (precision, real-time and cost). The final ranking has two components, the original ranking and the new ranking function generated with non-functional parameters, this function gives more weight to the services that have the precision as a non-functional parameter. As result we obtain the new reordered ranking (Table VI).

It is worth noting that the words are stored as terms of service interface, in order to speed up the fulfillment of future requests.

D. Retrieval-based Association

1) *Flow Ranking Generator*: The input for the final stage are the control terms that separate the service blocks retrieved in the previous module: , (Control) - before (Control) - and (Control) - if (Control) - not (Control) -, (Control). These terms are mapped to the most primitive workflow patterns described in [43]. The sequential pattern with the terms like *and*, *before* or *commas*, draws blocks one after the other in the order in which they are defined. The parallel pattern with terms like

or, draws the blocks in the same level and share the same root. The other pattern that we consider was the conditional pattern with terms like *if*, the pattern draws a block and bellow it, follows a parallel pattern to the others blocks.

As a result, the system creates a generic flow composed by five blocks, separated by control terms detected by the system:

Block 1 Separator Term: , (Control).

Block 2 Separator Term: before (Control).

Block 3 Separator Terms: and (Control) - if (Control) - not (Control).

Block 4 Separator Term: , (Control).

The result is shown in the Figure 5.

The reason why we choose to make a flowchart of basic components instead of standardized diagrams as UML, was because we consider that the flowchart is simpler to understand for end users. The generated graph has simple conventions: it represents the start and end nodes with a circle, the blocks of services that follow sequential or parallel patterns are represented as rectangles and the block with a conditional pattern is drawn as a rhomb.

2) *Flow Ranking Associator*: In this phase, the system replaces the generic blocks generated in the previous phase, for the block of retrieved services, with the exception of the start and end nodes. The result obtained is shown in the Figure 5. The conditional blocks, as the other blocks show the user the mapping of its queries in a generic flow, but the execution of the services is performed only in two ways: sequential and parallel. In the case of conditional blocks, the system executes the first service of the block in a sequential manner, the other blocks also execute the first service following one of the patterns. The execution of the services is performed by two different clients developed: one for web services, which selects the operation most similar to the service name and creates default values for the input; the other one for Telecommunication capabilities sends a Soap message to a Service Logic Execution Environment (SLEE) with the name of the service, and this executes the service with a default agent. In both cases, the responses are returned to the system. This way the user can see its logic request translated to a diagram and the response of the services. The services response is stored in a file indicating the name of the service, its response and the flow pattern that followed (sequential or parallel), this file and the graphic file are sent to the cell phone to be visualized by the user. This way, the user can retrieve services available on dynamic and varied context, see its functionality through the result after its execution and be useful to perform a formal composition process.

3) *Result Storage*: The terms are stored keeping a relationship with the services that were retrieved with the flow generated, given the possibility of future requests, in order to reduce processing time.

VII. EXPERIMENTAL STUDY

In this section, we present a perception of the quality of output according to the input, through two different evaluations of the first architecture phase (Analysis of Natural Language)

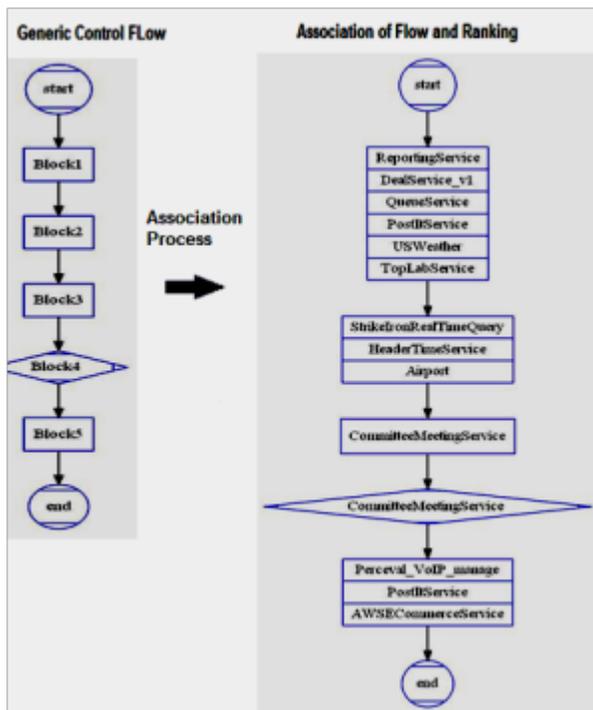


Fig. 5. Generic Control Flow

to determine also the viability of our proposal. Initially to the first evaluation, we took the most important module for the analysis of natural language requests; this module was *Named Entity Recognition* and was chosen, taking into account different features such as adaptation and relevance in the phase. The assessment was focused on measuring the accuracy of the system to classify entities that have already been defined by experts. Based on the measures defined in [44], [45]: *Precision* is the ratio of the number of correct words annotated as the entity by the system and the number of annotations of the entity given by the system, and the *Recall* is defined as the ratio between the number of correct words annotated as the entity by the system and the number of annotations of words as a specific entity given by the user. To develop the evaluation was used a corpus, obtained from different descriptions of the Google Play services. We use as a search parameter key words that were on the lists of the *Gazetter*. A Java program was implemented, to create the corpus from the android services description. The program removes useless characters leaving 7355 words.

The *Behavior* entities considered are: *message*, *call*, *maps*, *email*, *music*, *send*, *video*, *messenger* and *sms*; the *Input/Output* entities considered are: *location*, *person*, *number* and *time*; and the *Control* entities: *before*, *however*, *if*, *or*, and *while*.

With this in mind, experts who evaluated the system manually attributed annotations to the entities that describe the service interface (Input-Output, Behavior). Such assignments were performed according to the types of entities or words specified. The results for the NER evaluation are shown in

Table X, where: *Entity Made by the System* (EMS), *Correct Entity by the System* (CES) and *Entity Given by the User* (EGU) The second one is a general evaluation about user's

TABLE VII
NER EVALUATION RESULTS

Entity	Precision	Recall	EMS	CES	EGU
message	1	0,67	4	4	6
call	1	1	9	9	9
maps	1	1	14	14	14
email	1	0,67	6	6	9
music	1	0,89	17	17	19
send	1	0,94	18	18	19
video	1	0,52	11	11	21
messenger	0,9	0,36	10	9	25
sms	0,74	1	23	17	17
location	0,17	0,01	6	1	78
person	0,06	1	117	7	7
number	0,62	1	141	88	88
time	0,47	0,19	38	18	92
before	1	1	3	3	3
however	1	1	1	1	1
if	1	0,67	4	4	6
or	0,97	1	39	38	38
while	1	1	2	2	2

satisfaction, unlike to the *precision* and *recall* evaluation presented before. At first, we collect a set of forty five (45) requests in natural language from several Web users, collected from a survey that asked them, about the possible requests they would do to a retrieval service system in their mobile devices. In this vein, is important to consider that system will only understand readable and meaningful requests; however, in the case where a meaningless or misspell sentences be entered, the system will try to process them, in order to deliver consistent results to the user's request. All requests were categorized in high, medium and low scope, i.e., according to its content. *Low scope* represents requests without sense or without real services which cannot be implemented because its complexity (e.g., How do I fix this problem?), *medium scope* refers to incomplete or ambiguous service requests, which can be executed, but not in an ideal way (e.g., How do I get to "x site"?), and finally *high scope* request, are those which contains an adequate and reliable information of services (as the request shown in the example given). From former categories were selected randomly eight (8) answers, which became in input's system. To each request was obtained a set of properties and characteristics (e.g., word's grammatical category, number of identified services, etc.) which were being given by the system. To the evaluation, two important aspects were considered. In one side, we looked at classification of input words, organized in these groups: *Behavior*, *Input/Output* and *Control*. In the other side, we review the number of services identified and the words included in them.

Once taking all output system, ten human experts, with high knowledge in this subject, evaluated the outputs obtained by system. Experts were selected from research group named GIT (Group of Telematic Engineering) from Cauca's University, who work in the area of telematic services.

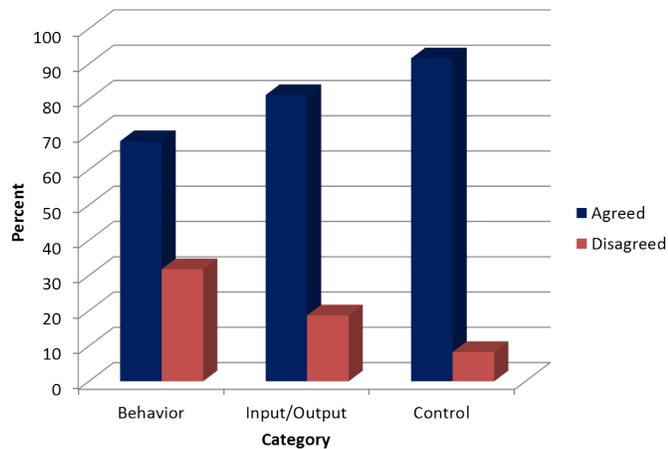


Fig. 6. Word classification satisfaction

Historical examples of this kind of evaluation methodologies are exemplified by "the fuma comparative evaluation of parsers of French (Abeillé), "the fi or competition of morphological analyzers for German" or "the Morpholympics (Hauser)" [39]. Thus, in order to determine trends of system's outputs, a survey was developed and applied to the above mentioned experts, in December of 2011. The survey applied to experts consisted of eight questions; each one represented the processing of only one request, and shows the classification of words from the sentences and the identification of the number of services. Two different sections can be recognized in the survey. The first one identifies words from input sentences that can be classified into *behavior*, *input/output* and *control* categories: *Control* words, *Behavior* words and *Input/Output* words. Experts have chosen, if they were agreed or disagreed with each proposed classification system. Percents of satisfaction are showed in Figure 6, where it is possible to show that higher percent of agreed selection was in Control category with a 91.6% and the lower one was presented in Behavior category with 68.1%. Also, in Input/Output category the 81.2% were agreed with the proposed selection. In the same way, experts were asked if they were agreed or disagreed with the number of services identified and the words detected in each service. Results in Figure 7, showed that 86.1% of experts considered that the number of services identified were optimums, while 13.8% not, and on the other hand 62.9% contemplate as adequate the classification of words in each service against a 37.0%. The other section of the survey, evaluates in general terms, whether the system is consider as excellent, good, regular or a bad system, according to all identification, selection and classifications parameters that were used. For this, the Figure 8 shows both outcomes, degree of satisfaction in number of services and words classification. In both cases was observed that most of the experts considered the system as good with a total of 41.6% and 59.7% respectively and only a minority considered a bad system with 2.7% and 1.3%. From this study we are able to determine if word's

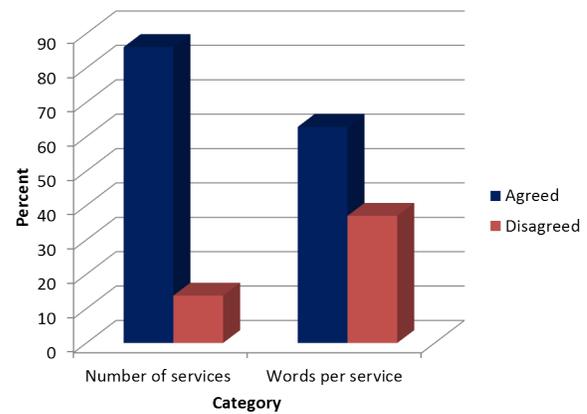


Fig. 7. Service classification satisfaction

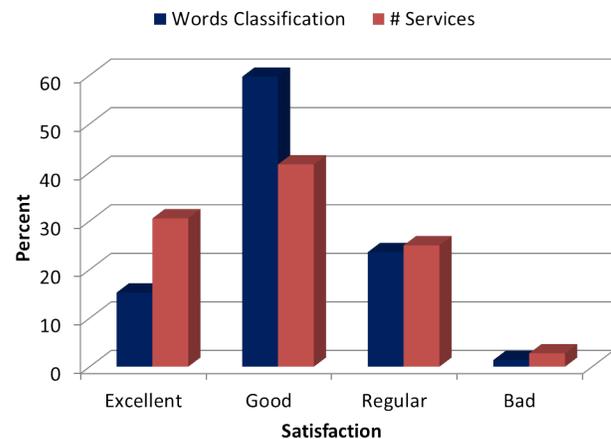


Fig. 8. General Evaluation

classification was optimums or not. Thus, results showed that *Behavior* classification had the lowest level of acceptance by experts, demonstrating the need to improve all the set of words and rules that govern and compose this category. This also implies that for subsequent tasks, specifically to development of matching phase, we must establish a higher percent of weigh to all *Input/Output* words, because they obtained better acceptance than *behavior*. Regarding identification of *Control* words, the percent of agreed is high, ensuring that control flow will be correctly established. In general terms the categorization of a good system by majority of experts, shows the adequate performance of the system, although can be improved in some aspects like rules definition, Gazetters lists, patterns and others related to increase exactitude levels.

VIII. CONCLUSION

This study presented a novel approach which facilitates the service retrieval process in converged environments, reducing complexity and habitual restrictions on incoming requests. Thereby, inexperienced users can express their needs in natural language, unlike other solutions that use templates restricting

user's expression. The system also takes into account from user request, functional and non-functional parameters, to enrich and give higher accuracy to retrieval service process.

The proposed system, processes requests made from mobile devices, and delivers a generic control flow as output, which contains the most suitable services to the users. From user's request analysis, we obtained a set of words classified in three different groups: Behavior, Control and Input/Output, which are used into matching module, to compare with characterization services tags. In the same way the folksonomy-based matchmaking, allowed an improvement of the system, by adapting to dynamic vocabulary and reducing the one used by end users.

In the proposed experimental study, where was assessed the quality of the outputs system, trends among experts revealed good acceptance in most of evaluated aspects, all above 55%. However, the survey also showed that Behavior classification must be improved to increase performance and accuracy of service retrieval.

The study of diverse techniques and related works, showed that implementing natural language analysis for service selection process, is efficient since it enhances and facilitates the processes, reducing the time required, complexity and increasing the number of eligible users. On the other hand, the implementation of a recommendation phase and auto-complete module, facilitated the understanding and interaction with the user, since it leads to formal elements in an informal request, without establish limitations in capturing requirements.

The proposed semantic words enrichment, increased accuracy in syntactic matching process, because when users enter words in a request, often they do not match the terms that describe services in the repository. Finally, the inclusion of non-functional properties in the service retrieval process, allows obtain not only the most related services to requests users, but also ensures that these services are the most optimal in terms of availability, response time documentation and other non-functional parameters.

IX. FUTURE WORK

As a complementary work, we can consider the enhancement of natural language phase, considering a module with more specific rules, to obtain higher accuracy of retrieval words. Natural Language Analysis phase could be also improved adding a module which detects meaningless request or incoherent and misspelled requests, to filter them and avoid the whole retrieval process of the system. As future work, the prototype can be extended to other language like Spanish, German, Mandarin, etc. Also the number of services and their respective descriptions (Functional and Non-functional properties) can be expanded involving other types of repositories which can enrich the current records.

As shown in the description, the system only executes services in an isolated manner, so for future work could be consider the execution of services taking into account the interfaces of the services such as, input/output features, respecting the order indicated in the generic control flow.

The generation of control flow, can be improved taking into account service interfaces. Finally for future work can be established and developed the other phases that composed the architecture as recommender phase, matching phase and association phase, to complete all functionality for which it was designed.

ACKNOWLEDGMENT

The authors would like to thank Universidad del Cauca, COLCIENCIAS and TelComp2.0 Project for supporting the Research of Camilo Pedraza and Julián Zúñiga

REFERENCES

- [1] E. C. Pedraza, J. A. Zuniga, L. J. Suarez Meza, and J. C. Corrales, "Automatic service retrieval in converged environments based on natural language request," in *SERVICE COMPUTATION 2011*, ser. 978-1-61208-152-6, Rome, Italy, September 25 2011, pp. 52–56.
- [2] F.-C. Pop, M. Cremene, M.-F. Vaida, and M. Riveill, "On-demand service composition based on natural language requests," in *Sixth International Conference on Wireless On Demand Network Systems and Services.*, ser. WONS'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 41–44. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1688899.1688905>
- [3] A. Bandara, "Semantic description and matching of services for pervasive environments," Ph.D. dissertation, University of Southampton, 2008. [Online]. Available: <http://eprints.ecs.soton.ac.uk/16403/>
- [4] ITU-T, "Series y: Global information infrastructure, internet protocol aspects and next-generation networks," INTERNATIONAL TELECOMMUNICATION UNION, Tech. Rep., 2001.
- [5] D. Moro, D. Lozano, and M. Macias, "Wims 2.0: Enabling telecom networks assets in the future internet of services," in *Proceedings of the 1st European Conference on Towards a Service Based Internet*, ser. ServiceWave '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 74–85.
- [6] ITU-T, "Converged services framework functional requirements and architecture," INTERNATIONAL TELECOMMUNICATION UNION, Tech. Rep., 2006.
- [7] J. M. E. Carlin and Y. B. D. Trinugroho, "A flexible platform for provisioning telco services in web 2.0 environments," in *Fourth International Conference on Next Generation Mobile Applications, Services and Technologies*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 61–66. [Online]. Available: <http://dx.doi.org/10.1109/NGMAST.2010.23>
- [8] G. Bond, E. Cheung, I. Fikouras, and R. Levenshteyn, "Unified telecom and web services composition: problem definition and future directions," in *Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications*, ser. IPTComm '09. New York, NY, USA: ACM, 2009, pp. 13:1–13:12. [Online]. Available: <http://doi.acm.org/10.1145/1595637.1595654>
- [9] S. Hagemann, C. Letz, and G. Vossen, "Web service discovery - reality check 2.0," in *Proceedings of the Third International Conference on Next Generation Web Services Practices*, ser. NWESP '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 113–118. [Online]. Available: <http://dx.doi.org/10.1109/NWESP.2007.31>
- [10] S. Kirati, "A demonstration on service compositions based on natural language request and user contexts," Master's thesis, Norwegian University of Science and Technology (NTNU), Department of Telematics, June 2008.
- [11] E. Al-Masri and Q. H. Mahmoud, "Discovering the best web service: A neural network-based solution," in *SMC*. IEEE, 2009, pp. 4250–4255.
- [12] A. V. Riabov, E. Boillet, M. D. Feblowitz, Z. Liu, and A. Ranganathan, "Wishful search: interactive composition of data mashups," in *Proceedings of the 17th international conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 775–784. [Online]. Available: <http://doi.acm.org/10.1145/1367497.1367602>
- [13] K. Bischoff, C. S. Firan, W. Nejdl, and R. Paiu, "Can all tags be used for search?" in *Proceedings of the 17th ACM conference on Information and knowledge management*, ser. CIKM '08. New York, NY, USA: ACM, 2008, pp. 193–202. [Online]. Available: <http://doi.acm.org/10.1145/1458082.1458112>

- [14] Apple. (2011, November) Getting to know siri. [Online]. Available: http://media.wiley.com/product_data/excerpt/80/11182992/1118299280-40.pdf
- [15] (2012) iphone user guide. [Online]. Available: http://manuals.info.apple.com/en_US/iphone_user_guide.pdf
- [16] IBM. (2011, August) What is watson? [Online]. Available: <http://www-03.ibm.com/innovation/us/watson/what-is-watson/index.html>
- [17] A. Bosca, F. Corno, G. Valetto, and R. Maglione, "On-the-fly construction of web services compositions from natural language requests." *JSW*, vol. 1, no. 1, pp. 40–50, 2006.
- [18] C. Christophe, "Specification of pro-active service infrastructure for attentive services," SPICE, Tech. Rep., 2007.
- [19] S. Angeletou, "Semantic enrichment of folksonomy tagspaces," in *Proceedings of the 7th International Conference on The Semantic Web*, ser. ISWC '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 889–894. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88564-1_58
- [20] M. Magableh, "A generic architecture for semantic enhanced tagging systems," Ph.D. dissertation, Montfort University, 2011.
- [21] G. Maciej, C. Giacomo, P. Marcin, and G. Maria, "Wscolab: Structured collaborative tagging for web service matchmaking," in *WEBIST (1)*, J. Filipe and J. Cordeiro, Eds. INSTICC Press, 2010, pp. 70–77.
- [22] S. E. Antonio Javier, "Open platform for user-centric service creation and execution," Telefónica I+D (ES), University of Valladolid (ES), Davidov, (BG), Ericsson (ES), Huawei (CN), IRIS (IT), JBoss (CH), Alcatel (DE), NEC (PT), Politecnico di Torino (IT), Portugal, Telecom Inovação (PT), Telecom Italia (IT), University of Madrid (ES), Tech. Rep., 2008.
- [23] C. Christophe, "Service platform for innovative communication environment," France Telecom (F), Alcatel (F), DoCoMo Communications Laboratories Europe (D), Telefonica (ESP), Telecom Italia (I), Telenor (NOR), Siemens (D,A), Ericsson (NL), Nokia (FIN), Stichting Telematica Instituut (NL), NEC Europe (UK), Bull (F), Fraunhofer FOKUS (D), University of Kassel (D), Alma Consulting Group (F), University of Brussels (B), IRIS (I), Neos (I), University of Surrey (UK), Norwegian University of Science and Technology (NOR), Politecnico di Torino (I), Telekomunikacja Polska (POL), Tech. Rep., 2008.
- [24] N. Jorg, L. Klostermann, F. Ioannis, S. Ulf, d. R. Frans, and O. Ulf, "Ericsson composition engine, next-generation in." Ericsson, Tech. Rep., 2009.
- [25] ITU-T, "Enhanced telecom operations map (etom) the business process framework;" INTERNATIONAL TELECOMMUNICATION UNION, Tech. Rep., 2004.
- [26] H. Jiejun, "A practical approach to the operation of telecommunication services driven by the tmf etom framework," Master's thesis, Universitat Politècnica de Catalunya, 2009.
- [27] TmForum. (2010) Information framework (sid)in depth. TmForum. [Online]. Available: <http://www.tmforum.org/InformationFramework/6647/home.html>
- [28] Yahoo. (2012) Pipes. [Online]. Available: <http://pipes.yahoo.com/pipes/>
- [29] (2012) Flickr de yahoo. [Online]. Available: <http://www.flickr.com/Flic>
- [30] A. Maaradji, H. Hacid, R. Skraba, A. Lateef, J. Daigremont, and N. Crespi, "Social-based web services discovery and composition for step-by-step mashup completion," in *Proceedings of the 2011 IEEE International Conference on Web Services*, ser. ICWS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 700–701. [Online]. Available: <http://dx.doi.org/10.1109/ICWS.2011.122>
- [31] H. Cunningham, "Gate, a general architecture for text engineering," *Computers and the Humanities*, vol. 36, pp. 223–254, 2002, 10.1023/A:1014348124664. [Online]. Available: <http://dx.doi.org/10.1023/A:1014348124664>
- [32] Apache. (2010) Apache opennlp developer documentation. [Online]. Available: <http://opennlp.apache.org/documentation/manual/opennlp.html>
- [33] GATE. (2011) Part-of-speech tags used in the hepple tagger. [Online]. Available: <http://gate.ac.uk/sale/tao/splitap7.html>
- [34] (2011) Developing language processing components with gate version 6. [Online]. Available: <http://gate.ac.uk/sale/tao/splitap7.html>
- [35] N. Iulia, "Desambiguación semántica automática," Ph.D. dissertation, University of Barcelona, 2002.
- [36] M. Lesk, "Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone," in *Proceedings of the 5th annual international conference on Systems documentation*, ser. SIGDOC '86. New York, NY, USA: ACM, 1986, pp. 24–26. [Online]. Available: <http://doi.acm.org/10.1145/318723.318728>
- [37] Snowball. (2011) Stemming algorithms. [Online]. Available: <http://snowball.tartarus.org/>
- [38] W. B. Frakes, "Term conflation for information retrieval," in *Proceedings of the 7th annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '84. Swinton, UK, UK: British Computer Society, 1984, pp. 383–389. [Online]. Available: <http://dl.acm.org/citation.cfm?id=636805.636830>
- [39] D. A. Hull, "Stemming algorithms: a case study for detailed evaluation," *J. Am. Soc. Inf. Sci.*, vol. 47, no. 1, pp. 70–84, 1996.
- [40] P. Mika, "Ontologies are us: A unified model of social networks and semantics," *Web Semant.*, vol. 5, no. 1, pp. 5–15, Mar. 2007. [Online]. Available: <http://dx.doi.org/10.1016/j.websem.2006.11.002>
- [41] D. Laniado, D. Eynard, and M. Colombetti, "A semantic tool to support navigation in a folksonomy," in *Proceedings of the eighteenth conference on Hypertext and hypermedia*, ser. HT '07. New York, NY, USA: ACM, 2007, pp. 153–154. [Online]. Available: <http://doi.acm.org/10.1145/1286240.1286282>
- [42] Seekda. (2012) Web service search. [Online]. Available: <http://webservices.seekda.com/search>
- [43] P. Ahana, "Workflow : Patterns and specifications," Master's thesis, Indian Institute of Technology, 2007.
- [44] P. Patrick, C. Stéphane, and H. Lynette, "Principles of evaluation in natural language processing," *TAL*, vol. 48, p. 23, 2007.
- [45] M. King, "Evaluating natural language processing systems," *Commun. ACM*, vol. 39, no. 1, pp. 73–79, January 1996. [Online]. Available: <http://doi.acm.org/10.1145/234173.234208>

Ad hoc Iteration and Re-execution of Activities in Workflows

Mirko Sonntag, Dimka Karastoyanova

Institute of Architecture of Application Systems
University of Stuttgart, Universitaetsstrasse 38
70569 Stuttgart, Germany

sonntag@iaas.uni-stuttgart.de, karastoyanova@iaas.uni-stuttgart.de

Abstract—The repeated execution of workflow logic is usually modeled with loop constructs in the workflow model. But there are cases where it is not known at design time that a subset of activities has to be rerun during workflow execution. For instance in e-Science, scientists might have to spontaneously repeat a part of an experiment modeled and executed as workflow in order to gain meaningful results. In general, a manually triggered ad hoc rerun enables users reacting to unforeseen problems and thus improves workflow robustness. It allows natural scientists steering the convergence of scientific results, business analysts controlling their analyses results, and it facilitates an explorative workflow development as required in scientific workflows. In this paper, two operations are formalized for a manually enforced repeated enactment of activities, the iteration and the re-execution. The focus thereby lies on an arbitrary, user-selected activity as a starting point of the rerun. Important topics discussed in this context are handling of data, rerun of activities in activity sequences as well as in parallel and alternative branches, implications on the communication with partners/services and the application of the concept to workflow languages with hierarchically nested activities. Since the operations are defined on a meta-model level, they can be implemented for different workflow languages and engines.

Keywords—*workflow ad hoc adaptation; iteration; re-execution; service composition*

I. INTRODUCTION

Imperative workflow languages are used to describe all possible paths through a process. On the one hand, this ensures the exact execution of the modeled behavior without deviations. On the other hand, it is difficult, if not impossible, to react to unforeseeable and/or un-modeled situations that might happen during workflow execution, e.g., exceptions, changes in regulations in business processes, etc. This is the reason why flexibility features of workflows were identified as essential for the success of the technology in real world scenarios [2, 3, 4]. In [5], four possible modifications of running workflows are described as advanced functions of workflow systems: the deletion of steps, the insertion of intermediary steps, the inquiry of additional information, the iteration of steps.

This paper focusses on the iteration of steps. Usually, iterations are explicitly modeled with loop constructs. However, not all eventualities can be accounted for in a process model prior to runtime. Imagine a process with an

activity to invoke a service. At runtime, the service may become unavailable. The activity and hence the process will fail, leading to a loss of time and data, if the underlying service middleware cannot tackle the problem with failing services. An ad hoc operation to rerun the activity (maybe with modified input parameters) could prevent this situation.

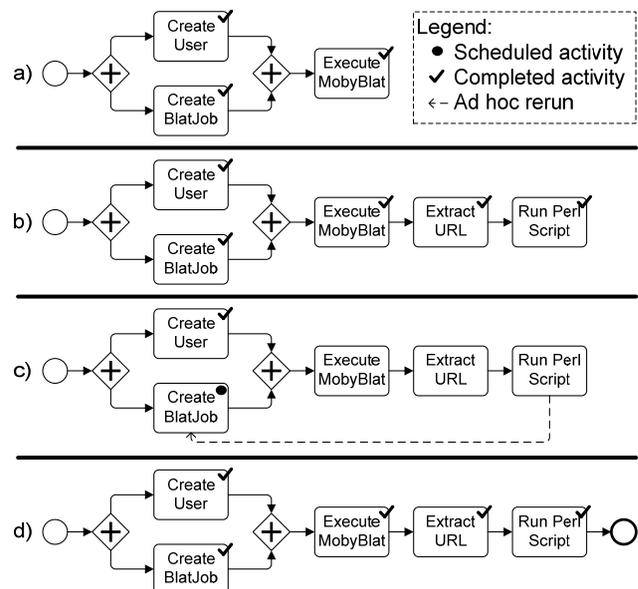


Figure 1. Example for the flexible development of a scientific workflow (borrowed from [6])

The repetition of workflow logic is not only meaningful for handling faults. In the area of scientific workflows, the result of scientific experiments or simulations is not always known a priori [6, 7, 8, 9]. Scientists may need to take adaptive actions during workflow execution. In this context, rerunning activities is basically useful to enforce the convergence of results, e.g., redo the generation of a Finite Element Method (FEM) grid to refine a certain area in the grid, repeat the visualization of results to obtain an image with focus on another aspect of a simulated object, enforce the execution of an additional simulation time step. A simplified example for an explorative development of a scientific workflow is given in Figure 1 (the example is borrowed from [6]). In this scenario, a scientist wants to perform a search for a DNA sequence in a particular genome using a Blat Web service. He models a workflow with three

tasks and puts them in the order presented in (Figure 1a): “Execute MobyBlat” invokes the scientific Web service; “Create User” creates the input for MobyBlat containing a specific database to search in and providing user credentials; “Create BlatJob” configures the search operation and contains the DNA sequence to search for in the selected genome. The scientist runs this workflow (Figure 1a). He takes a look at the result of the MobyBlat service and discovers that the result format is a MOBY-S XML object. The result object contains a URL to the final result, the Blat report. In order to download the report he adapts the running workflow by appending two additional tasks: “Extract URL” gets the URL to the Blat report out of the MOBY-S XML object; “Run Perl Script” starts a Perl script that downloads the report (Figure 1b). The scientist inspects the downloaded report and recognizes that it has an inappropriate format. Hence, he reruns the workflow from the “Create BlatJob” task on (Figure 1c). In this second execution, he configures the BlatJob so that the Blat Web service delivers the expected format (Figure 1d). With this the scientist finishes the development of this scientific workflow in an iterative manner. The ad hoc adaptation of the workflow and the ad hoc rerun operation prevent a loss of data, time and money compared to a restart of the complete workflow and hence the creation of a new workflow instance. This is especially the case for long-running (scientific) workflows. In the example, the scientist does not have to provide the input for the “Create User” task again. There are other scenarios where the visualization of scientific results is repeated several times with different parameters without a need to rerun the complete long-running scientific simulation.

A significant number of approaches exist for enabling the repetition of activities in workflows. Existing approaches use modeling constructs (e.g., loops, BPEL retry scopes [10]), workflow configurations (e.g., Oracle BPM [11]), or an automatic rerun of faulted activities (e.g., Pegasus [12]) to realize the repeated execution of workflow parts. An approach for the ad hoc repetition of workflow logic with an *arbitrary starting point* that was user-selected at runtime is currently missing in industrial workflow engines and insufficiently addressed in research. Such functionality is useful in both business and scientific workflows. In business workflows it can help to address faulty situations, especially those where a rerun of a single faulted activity (usually a service invocation) is insufficient, or changes in the control logic needed to address new requirements. In scientific workflows it is one missing puzzle piece to enable explorative workflow development [7, 8] and to control and steer the convergence of results.

This paper therefore focusses on enabling the rerun of activities in workflows from arbitrary points in the workflow model. Two operations on workflow instances are formalized to enforce the repetition of workflow logic: the *iteration* works like a loop that reruns a number of activities; the *re-execution* undoes work completed by a set of activities with the help of compensation techniques prior to the repetition of the same activities. The operations are defined on the level of the workflow meta-model. Thus, the operations can be implemented in different workflow

languages and engines. Problems such as data handling issues, the communication with partners, or how the concept can be applied in workflow languages with block structures are identified and discussed. This paper is a logical continuation of our work presented in [1]. Note that the terms “workflow” and “process” are used interchangeably.

The rest of the paper is organized as follows. Section II shows the workflow meta-model used in this work. Section III describes the *iterate* and *re-execute* operations. Section IV addresses data handling issues and Section V applies the approach in more complex workflow graphs including parallel and alternative branches. Section VI discusses implications of the approach on message-receiving and message-sending activities, on reruns within loops, and on reruns in workflow languages with block structures. Section VII shows how users interact with a workflow system that implements the manually enforced repetition of workflow logic. Section VIII is devoted to the prototypical implementation of the concepts based on BPEL. Section IX presents work related to the research topic of this paper. Finally, Section X concludes the paper.

II. META-MODEL

The workflow meta-model used in this paper is based on the one provided in [5]. It is adapted where appropriate in order to accommodate the aspects needed to describe the repeated execution of workflow logic. A process model is considered a directed, acyclic graph (Figure 2). The nodes are tasks to be performed (i.e., activities). The edges are control connectors (or links) and prescribe the execution order of activities. Data dependencies are represented by variables that are read and written by activities. In the description of the meta-model $\wp(S)$ is used as the power set of a given set S .

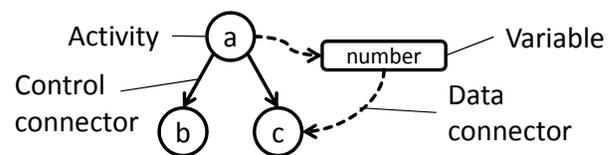


Figure 2. Example for a process model

A. Modeling

A workflow model can be expressed with the help of sets for the different workflow elements defined in the following.

Definition “Variables, V ”. The set of variables defines all variables of a process model:

$$V \subseteq M \times S \quad (1)$$

M is the set of names and S denotes the set of data structures. Each $v \in V$ has assigned a finite set of possible values, its domain $\text{DOM}(v)$ [5].

Definition “Activities, A ”. Activities are functions that perform tasks. The set of all activities of a process model is

$$A \subseteq M \times C. \quad (2)$$

C is the set of all conditions in a process model and is used here as join condition for an activity. If $j \in C$ evaluates to `true` at runtime, the activity is instantiated and scheduled (i.e. the navigator is going to execute the activity). Variables can be assigned to activities via an input variable map

$$i: A \mapsto \wp(V) \quad (3)$$

and an output variable map

$$o: A \mapsto \wp(V). \quad (4)$$

Input variables may provide data to activities and activities may write data into output variables. Furthermore, compensating activities that undo the effects of an activity can be assigned by a compensate activity map

$$c: A \mapsto A. \quad (5)$$

This map reflects the concept that activities can be considered as pairs consisting of an activity and its compensating activity. The idea is geared towards the approach of sagas [13]: The workflow can thereby be considered as a long-lived transaction implemented as saga, i.e. as non-atomic transaction that consists of a sequence of atomic sub-transactions T_1, \dots, T_n ; an activity $a \in A$ with a compensating activity is like an atomic sub-transaction T_j in a saga, and the compensating activity $c(a)$ can be compared to a compensating transaction C_j .

Definition “Links, L ”. The set that denotes all control connectors/links in a process model is

$$L \subseteq A \times A \times C. \quad (6)$$

Each link connects a source with a target activity. Its transition condition $t \in C$ determines at runtime if the link is followed. Two activities can be connected with at most one link (i.e., links are unique).

Definition “Process Model, G ”. A process model is a directed acyclic graph denoted by a tuple

$$G = (m, V, A, L) \quad (7)$$

with a name $m \in M$.

B. Execution And Navigation

For the execution of a process model, a new process instance of that model is created, activities are scheduled and performed, links are evaluated, and variables are read and written. These tasks (i.e., the navigation) are conducted according to certain rules. The component of a workflow

system that supervises workflow execution and that implements these rules is called the *navigator*. The notion of time in the meta-model is reflected with ascending natural numbers. Each process instance possesses its own timeline. At time $0 \in \mathbb{N}$ a process is instantiated. Each navigation step increases the time by 1. In the following, the navigation rules that are most important for this work are presented.

If an activity is executed, an activity instance is created with a new unique id. If the same activity is executed again (e.g., because it belongs to a loop), another activity instance is created with another id. The same holds for links and link instances. A new id can be generated with the function `newId()` that delivers an element of the set of ids, ID .

Process, activity and link instances are considered sets of tuples. This allows navigating through a process by using set operations. Navigation steps are conducted by creating new tuples and adding them to sets (instantiation of an activity/a link) or by deleting tuples from sets and adding modified tuples (to change the state of existing activity/link instances).

Definition “Variable Instances, V^I ”. Variable instances provide a concrete value c for a variable v (i.e., an element of its domain) at a point in time t . The finite set of variable instances is denoted as

$$V^I = \{(v, c, t) \mid v \in V, c \in \text{DOM}(v), t \in \mathbb{N}\}. \quad (8)$$

The set of all possible variable instances is V^I_{all} .

Definition “Activity Instances, A^I ”. The set of activity instances is denoted as

$$A^I = \{(id, a, s, t) \mid id \in ID, a \in A, s \in S, t \in \mathbb{N}\}. \quad (9)$$

At a point in time t an activity instance $a \in A^I$ has an execution state $s \in S = \{S, E, C, F, T, \text{COMP}, D\}$. The meaning of the states is as follows:

- S , scheduled: The activity is in the execution queue of the navigator but not yet running. The navigator is going to execute the activity in future.
- E , executing: The activity is running.
- C , completed: The activity was successfully executed.
- F , faulted: A fault happened during activity execution.
- T , terminated: Abortion of a scheduled or executing activity by the user.
- COMP , compensated: The compensation activity $c(\text{model}(a))$ was executed successfully for a completed activity.
- D , dead: The activity is located in a dead path, i.e., a path with links evaluated to `false`. It was neither scheduled nor executed.

The function `model(a)` for an activity instance $a = (id, a, s, t) \in A^I$ delivers its activity model a . Note that there is at most one instance of an activity in A^I . That way A^I exactly

reflects the process instance state in the current iteration. There is no influence by activity states from former iterations. While this condition is inherent for workflows without loops, it must be explicitly ensured by the navigator component of the workflow engine for more complex workflow executions including loops or manual ad hoc reruns of activities (in the focus of this work).

In the following, three sets are defined that help to capture the state of a process instance and that are used to navigate through a process model graph. These sets extend the meta-model described in [5].

Definition “Active Activities, A^A ”. The finite set of active activities A^A contains all activity instances that are scheduled or currently being executed:

$$A^A \subseteq A^I, \forall a \in A^A: \text{state}(a) \in \{S, E\}. \quad (10)$$

The function $\text{state}(a)$ for an activity instance $a = (\text{id}, a, s, t) \in A^I$ returns its current state $s \in S$.

Definition “Finished Activities, A^F ”. The finite set of finished activities A^F contains all activity instances that are completed, faulted, terminated, or dead:

$$A^F \subseteq A^I, \forall a \in A^F: \text{state}(a) \in \{C, F, T, D\}. \quad (11)$$

This set is needed to assure a preconditions for the repetition of activities and for the compensation of already completed work. Note that compensated activities are not part of A^F because their effects are undone.

Definition “Evaluated Links, L^E ”. The finite set of evaluated links L^E contains link instances whose transition condition is already interpreted. Link instances refer to the instantiated link l , have a truth value c for the evaluated transition condition and an execution time t :

$$L^E = \{(l, c, t) \mid l \in L, c \in \{\text{true}, \text{false}\}, t \in \mathbb{N}\}. \quad (12)$$

Note that each link has at most one link instance in L^E for one process instance. If a link is evaluated repeatedly (e.g., due to a loop or a manual ad hoc rerun), the old link instance must be removed from L^E . This is ensured by the navigator component of the workflow engine in order to prevent an interference of link instances of different workflow iterations. Note that the set of evaluated links is usually not part of the context of a workflow instance in typical workflow engines (cf. [5, 14, 15]). The link state (i.e., the truth value c) is only important to evaluate the join condition of the link’s target activity and can be thrown away afterwards. In this work, the context of process instances is extended by storing the truth value for all evaluated links because it is needed for a correct join behavior if join activities are rerun. The set L^E is very similar to the markings of control connectors known from ADEPT [3, 16].

Definition “Wavefront, W ”. The set of all active activities and evaluated links, for which the target activity is not yet scheduled, is called the process instance’s wavefront

$$W = A^A \cup L^A \quad (13)$$

with $L^A \subseteq L^E, \forall l \in L^A: \nexists a \in A^A \cup A^F: \text{target}(\text{model}(l)) = \text{model}(a)$. The function $\text{model}(l)$ for a link instance $l = (l, c, t) \in L^E$ delivers its link model l . The function $\text{target}(l)$ for a link $l = (a, b, c) \in L$ returns its target activity b .

Definition “Process Instance, p_g ”. An instance for a process model g is now defined as a tuple

$$p_g = (V^I, A^A, A^F, L^E). \quad (14)$$

TABLE I. THE NAVIGATION EXAMPLE SHOWS HOW THE WORKFLOW ENGINE EXECUTES A WORKFLOW INSTANCE BY SET OPERATIONS.

Time	V^I	A^A	A^F	L^E
1	{(number, 100, 1)}		{}	{}
2	{(number, 100, 1)}	{(382, a, S, 2)}	{}	{}
3	{(number, 100, 1)}	{(382, a, E, 3)}	{}	{}
4	{(number, 100, 1), (number, 101, 4)}	{(382, a, E, 3)}	{}	{}
5	{(number, 100, 1), (number, 101, 4)}	{}	{(382, a, C, 5)}	{}
6	{(number, 100, 1), (number, 101, 4)}	{}	{(382, a, C, 5)}	{(a-b, true, 6)}
7	{(number, 100, 1), (number, 101, 4)}	{}	{(382, a, C, 5)}	{(a-b, true, 6), (a-c, false, 7)}
8	{(number, 100, 1), (number, 101, 4)}	{(383, b, S, 8)}	{(382, a, C, 5)}	{(a-b, true, 6), (a-c, false, 7)}
9	{(number, 100, 1), (number, 101, 4)}	{(383, b, E, 9)}	{(382, a, C, 5)}	{(a-b, true, 6), (a-c, false, 7)}
10

The set of all process instances is denoted as P_{all} . As navigation example consider the workflow model in Figure 2 and a corresponding workflow instance in Table 1. Say the workflow is instantiated and the variable $\text{number} \in V$ is initialized with 100 (time step 1). Then, activity $a \in A$ is scheduled (2) and executed (3). Suppose activity a models

the invocation of a program that increases a given number by 1. The variable “number” is used as input value for this operation and is hence updated (4), i.e., the tuple representing the variable instance is substituted. Activity a completes and its corresponding instance tuple is deleted from A^A and a new tuple containing the new activity instance state with increased time step is added to A^F (5). Now the

navigator evaluates the transition condition of the links a-b and a-c; a-b's condition evaluates to `true` (6), a-c's to `false` (7). As a consequence, the target activity of a-b is scheduled and executed (8 and 9). Note that even though the navigator manipulates the tuples, all these actions are recorded in the audit trail [5].

III. ITERATION AND RE-EXECUTION

Based on the meta-model described above the repeated execution of workflow parts is described in this section. As already proposed in [10], two repetition operations are thereby distinguished. The first operation, iteration, reruns workflow parts without taking corrective actions or undoing already completed work. The second operation, re-execution, resets the workflow context and execution environment with compensation techniques prior to the rerun (e.g., deallocating reserved computing resources, undoing completed work).

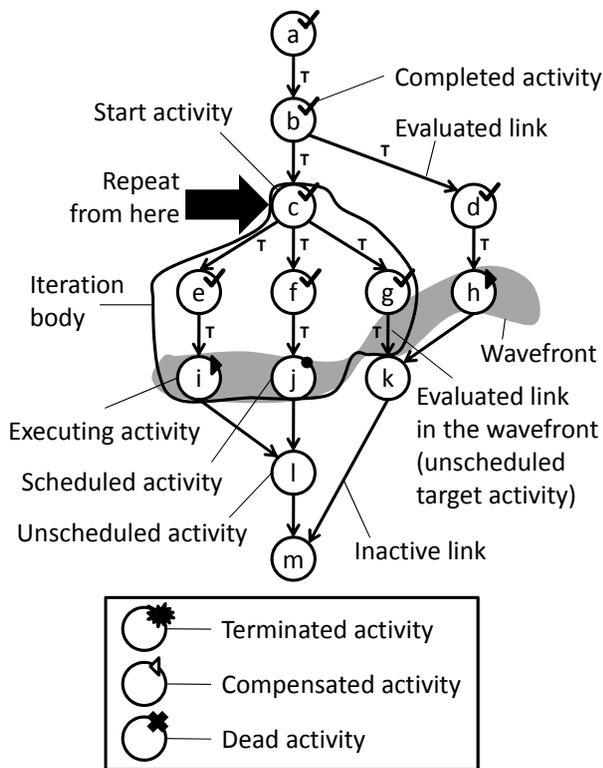


Figure 3. Example of a process instance

Before going into the details of the iteration of workflow parts several important terms are introduced (see Figure 3). The point from where a workflow part is executed repeatedly is denoted as the *start activity* (activity c in the figure). The start activity is chosen manually by the user/scientist at workflow runtime. The workflow logic from the start activity to those active activities and active links that are reachable from the start activity are called *iteration body* (activities c, e, f, g, i, j, the links in between and link g-k). The iteration body is the logic that is executed repeatedly. Note that activities/links reachable from the iteration body but not in

the iteration body are executed normally when the control flow reaches them (e.g., activities k and l).

For the iteration/re-execution of logic it is important to avoid race conditions, i.e., situations where two or more distinct executions of one and the same path are running in parallel. These situations can occur in cyclic workflow graphs or can be introduced by the manual rerun of activities this work deals with. For example, if the repetition is started from activity c in Figure 3, a race condition emerges because activities i and j on the same path are still running: activity l could be started if i and j complete while a competing run is started at c. There are two ways to avoid race conditions in this scenario. Firstly, the workflow system can wait until the running activities in the iteration body are finished without scheduling any successor activities (here: l). The rerun is triggered afterwards. Secondly, running activities in the iteration body can be terminated and the rerun can start immediately. A workflow system should provide both options to the users. In some cases it is meaningful to complete running work prior to the rerun (e.g., to reach a consistent system state), in other cases an abortion is a better choice (e.g., because the result of running work is unimportant or the activities being executed are long-running). This has to be decided on a per-case-basis by the user. In the rest of the paper the focus lies on the option "termination" since it is more complex and requires one step more than the option "wait for completion". However, "wait for completion" can be derived from the descriptions by omitting the explicit termination of activities in the examples.

Definition "Activities in Iteration Body". A function is needed that finds all activity instances in the iteration body of an activity in a given process instance. The function is useful for terminating active activities in the iteration body (or for waiting for their completion) to avoid race conditions and for resetting finished activities to avoid interference of activity execution states in different activity runs:

$$\text{activitiesInIterationBody}: A \times P_{\text{all}} \mapsto \wp(A^I) \quad (15)$$

Let $a \in A$ be an activity in process model g and $p_g \in P_{\text{all}}$ an instance of g . Then $\text{activitiesInIterationBody}(a, p_g) = \{a_1, \dots, a_k\}$, $a_1, \dots, a_k \in A^I \Leftrightarrow \forall i \in \{1, \dots, k\}: \text{model}(a_i)$ is reachable from activity a . An algorithm for the "activities in iteration body" function can be implemented by walking through the workflow graph beginning with activity a until the wavefront or an already visited activity is reached. The activity instance for each considered activity is stored. Since each activity is visited at most once, the algorithm is in $O(n)$, with n as the number of activities in the workflow model.

Race conditions can also occur if evaluated links in the iteration body remain in the process instance. In Figure 3, a race condition could appear as follows. If activity h completes and the link h-k is evaluated, the join condition of activity k could become true. Activity k would then be started although a competing execution of the same path

arises due to the repetition of activity c . That is why such links have to be found and reset, i.e., they are deleted from the set of evaluated links L^E .

Definition “Links in Iteration Body”. A function is needed that finds all evaluated links in the iteration body in a given activity and process instance:

$$\text{linksInIterationBody}: A \times P_{\text{all}} \mapsto \wp(L^E) \quad (16)$$

Let $a \in A$ be an activity of process model g and $p_g \in P_{\text{all}}$ an instance of g . Then $\text{linksInIterationBody}(a, p_g) = \{l_1, \dots, l_k\}$, $l_1, \dots, l_k \in L^E \Leftrightarrow \forall i \in \{1, \dots, k\}: \text{model}(l_i)$ is reachable from activity a . An algorithm for the “links in iteration body” function can be implemented by traversing the workflow graph starting from activity a . Each path has to be followed only until the wavefront or an already visited activity is reached. Since each link is visited at most once, the complexity of such an algorithm is in $O(n)$, where n is the number of activities in the workflow model.

A. Iteration

Parts of a workflow may be repeated without the need to undo any formerly completed work. A scientist may want to enforce the convergence of experiment results and therefore repeats some steps of a scientific workflow.

Definition “Iterate Operation”. The iteration is a function that repeats logic of a process model for a given process instance. A specified activity is the starting point of the operation. The input data elements for the iteration are either the current variable values or are loaded from a specified variable snapshot that belongs to the start activity.

$$\iota: A \times P_{\text{all}} \mapsto P_{\text{all}} \quad (17)$$

Let $a \in A$ be the start activity of the iteration and $p_{\text{in}_g}, p_{\text{out}_g} \in P_{\text{all}}$ two process instances. Here, p_{in_g} is the input for the ι operation and p_{out_g} is the resulting instance with changed state that is ready to start with the iteration. The pre-condition is that only already instantiated but no dead activities can be used as start activity:

$$\exists n \in A^A \cup A^F: \text{state}(n) \notin \{D\} \wedge \text{model}(n) = a. \quad (18)$$

This prevents (1) using the operation on dead paths and (2) jumping into the future of a process instance, which are both not a repetition of completed workflow logic.

Then $\iota(a, p_{\text{in}_g}) = p_{\text{out}_g}$, $p_{\text{in}_g} = (V_{\text{in}}^I, A_{\text{in}}^A, A_{\text{in}}^F, L_{\text{in}}^E)$ and $p_{\text{out}_g} = (V_{\text{out}}^I, A_{\text{out}}^A, A_{\text{out}}^F, L_{\text{out}}^E) : \Leftrightarrow$

1. $V_{\text{out}}^I = V_{\text{in}}^I$
2. $A_{\text{out}}^A = A_{\text{in}}^A \setminus \text{activitiesInIterationBody}(a, p_{\text{in}_g}) \cup \{(\text{newId}(), a, S, t)\}$, t is a new and youngest time step
3. $A_{\text{out}}^F = A_{\text{in}}^F \setminus \text{activitiesInIterationBody}(a, p_{\text{in}_g})$
4. $L_{\text{out}}^E = L_{\text{in}}^E \setminus \text{linksInIterationBody}(a, p_{\text{in}_g})$

The variables remain unchanged (1.). This reflects the case where the current variable values are taken as input for the iteration. All active successor activities from a are terminated, i.e., deleted from the set of running activities A^A (2.). All finished activities in the iteration body are reset, i.e., removed from the set of finished activities A^F (3.). All evaluated links in the iteration body are reset, i.e., their evaluation result is deleted from the set of evaluated links L^E (4.). The start activity is scheduled (added to the set of active activities with status scheduled, S) so that the workflow logic is repeated beginning with the start activity (2.). The join condition of the start activity is not evaluated again.

In the second case of the ι operation, a variable snapshot is loaded prior to the iteration. The loaded variable values are taken as input for the iteration:

$$1. V_{\text{out}}^I = V_{\text{in}}^I \cup \text{loadSnapshot}(b, p_{\text{in}_g}, e, V')$$

Here, $b \in A$ is the activity to load the snapshot for (the start activity or a predecessor thereof), $e \in \mathbb{N}$ is the execution number of b needed to select the correct snapshot instance, $V' \subseteq V$ is a subset of variables to be loaded from the snapshot. The complete definition of the function can be found in Section IV in (21). Snapshots are stored during process execution before each activity that modifies variables. A snapshot is uniquely addressed by its corresponding activity b and an execution number e . The latter is needed because there can be several snapshot instances for an activity—one for each activity execution. The subset of variables V' can be specified by the user to select particular variables that should be loaded from a snapshot. That means it is possible to load only a part of a snapshot. This is especially important for iterations in parallel paths. Variables that are not loaded from the snapshot have the same value as in the original process instance p_{in_g} . For more details about data handling see Section IV.

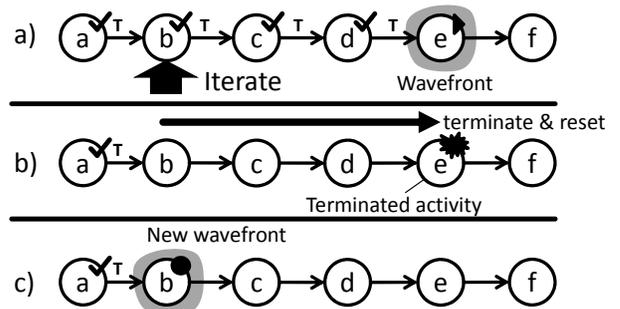


Figure 4. Iteration in a sequence of activities. The user requests the iteration of activities (a). The iteration body is reset and active activities are terminated (b). Finally, the start activity of the iteration is scheduled (c).

As an example for the ad hoc iteration of workflow logic consider Figure 4. There is a sequence of activities. Activity e is currently being executed. The user wants to iterate the workflow with activity b as start activity (Figure 4a). The path from b to the wavefront is traversed, visited links are reset (b - c , c - d and d - e in the example), and scheduled or

running activities are terminated (activity e), as shown in Figure 4b. Finally, a data snapshot is loaded (if requested by the user) and the start activity (b) is scheduled (Figure 4c).

B. Re-execution

It is also needed to repeat parts of a workflow as if they were executed for the first time. Completed work in the iteration body has to be reversed/compensated prior to the repetition. A scientist may want to retry a part of an experiment because something went wrong. But the execution environment has to be reset first (e.g. stateful services have to be set to their initial state, computing resources have to be released).

Algorithm “Compensate Iteration Body”. For the compensation of completed work in the iteration body an algorithm with the following signature is defined:

$$\text{compensateIterationBody}: A \times P_{\text{all}} \mapsto \wp(V_{\text{all}}^I) \quad (19)$$

The function compensates all completed activities of the iteration body in reverse execution order. It delivers the values of variables that were changed during compensation.

Let $a \in A$ be the start activity of the re-execution and $p \in P_{\text{all}}$ a process instance for the model of activity a. Then $\text{compensateIterationBody}(a, p) = \{v_1, \dots, v_k\}$ with $p = (V^I, A^A, A^F, L^E), v_1, \dots, v_k \in V_{\text{all}}^I$ works as follows (Note:

The function $\text{time}(f)$ with $f \in A^I$ delivers the time of the last state change of activity instance f):

```

function compensateIterationBody(a, p)
1   $V_{\text{result}} \leftarrow \emptyset$ 
2   $F = \{f \in A^F \mid \text{state}(f) == \text{completed} \wedge$ 
    $\text{model}(f) \text{ is reachable from } a\}$ 
3  while ( $|F| > 0$ ) do
4    if ( $|F| > 1$ ) then
5       $\exists m \in F: \forall n \in F, n \neq m:$ 
         $\text{time}(m) > \text{time}(n) \Rightarrow \text{execute}$ 
         $\text{compensating activity } c(\text{model}(m))$ 
6    else
7       $\exists m \in F \Rightarrow \text{execute compensating}$ 
         $\text{activity } c(\text{model}(m))$ 
8    end if
9     $F \leftarrow F \setminus \{m\}$ 
10   for each ( $v \in o(c(\text{model}(m)))$ ) do
11     if ( $\exists w \in V_{\text{result}}: \text{model}(w) = v$ ) then
12        $V_{\text{result}} \leftarrow V_{\text{result}} \setminus \{w\}$ 
13     end if
14      $V_{\text{result}} \leftarrow V_{\text{result}} \cup \{(v, c, t)\}$ , c is the
        new value of variable v, t is the
        timestamp of the assignment
15   end for
16 end while
17 return  $V_{\text{result}}$ 

```

A similar algorithm for the creation of the reverse order graph is also proposed in [17]. But the intention of the

Algorithm “Compensate Iteration Body” is to deliver the changed variable values as result of the compensation operation.

Definition “Re-execute Operation”. The re-execution is a function that repeats logic of a process model for a given process instance with a given activity as starting point. The input data for the re-execution is taken from a variable snapshot that belongs to the start activity or a predecessor of the start activity. The “re-execute” uses the “compensate” operation to reset already completed work in the iteration body.

$$\rho: A \times P_{\text{all}} \mapsto P_{\text{all}} \quad (20)$$

The start activity $a \in A$, the process instances $p_{\text{in}_g}, p_{\text{out}_g} \in P_{\text{all}}$, and the pre-condition are similar to the iterate operation. The difference is the calculation of V_{out}^I :

$$\rho(a, p_{\text{in}_g}) = p_{\text{out}_g} : \Leftrightarrow$$

1. $V_{\text{out}}^I = V_{\text{in}}^I \cup \text{compensateIterationBody}(a, p_{\text{in}_g})$
 $\cup \text{loadSnapshot}(b, p_{\text{in}_g}, t, V')$.

The variable values might be modified as a result of the compensation of completed work in the iteration body or by loading a data snapshot (1.). Note that the start activity for the re-execution is scheduled after the compensation is done and the snapshot is loaded.

An example for the re-execution of activities is given in Figure 5. In a sequence of activities, activity e is currently being executed. The user decides to re-execute the workflow from activity b (Figure 5a). The path from b to the wavefront (activity e) is followed. All links on this path are reset (links b-c, c-d, and d-e) and all activities in the wavefront reachable from b are terminated (activity e). Now, all completed activities in the iteration body are compensated in reverse execution order (Figure 5b). Note that only completed activities with an attached compensation activity can be compensated. Finally, a data snapshot is loaded and the start activity is scheduled (Figure 5c).

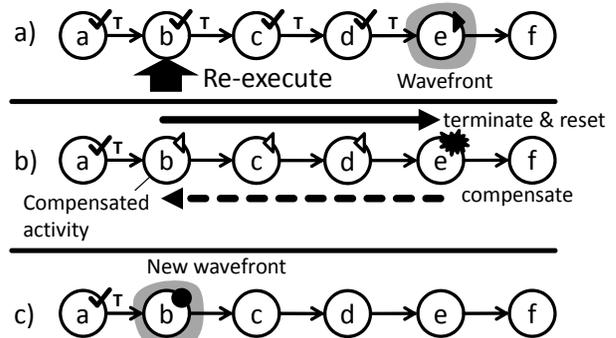


Figure 5. Re-execution in a sequence of activities. The user wants to re-execute activities (a). The iteration body is reset, active activities are terminated, and completed activities are compensated in reverse execution order (b). Finally, a data snapshot is loaded and the start activity for the re-execution is scheduled (c).

In practice, compensation of already completed work is not always possible. An invoked service must provide an operation to undo the results of a former request. For instance, a service with an operation to book a hotel room should also provide an operation to cancel the booking. The ρ operation relies on such compensation operations of services to conduct the compensation of already completed workflow logic in the iteration body. It is up to the person that models the considered workflow to integrate compensation logic in form of a compensating activity $c(a)$ for an activity a . This is a prerequisite for the correct and desired functionality of the re-execution.

IV. DATA HANDLING

For the repetition of workflow parts the handling of data is of utmost importance. Some of the questions that arise are: Where to store data that the former iteration has produced? What data should be taken as input for the next iteration? A mechanism is needed to store different values for same variables and to load appropriate/correct variable values for iterations. Variables might also be reset by the compensation of the iteration body as is done in the “re-execute” operation. This strongly depends on the compensation logic and invoked services. But it cannot be guaranteed that the former variable values are restored by the compensation. Hence, another mechanism is needed.

The desired functionality can be realized by saving snapshots of variables during workflow execution. Available workflow engines store an audit trail [5, 18] that contains, amongst others, values of variables changed by the successfully executed activities. However, the audit trail saves variables incoherently. The proposed snapshot mechanism contains only variables that are visible for a particular activity, their current values and a timestamp. The data footprint of snapshots can be minimized as follows: values for variables that did not change between two snapshots do not have to be stored again; pointers to the values of variables can be used to still be able to refer to them.

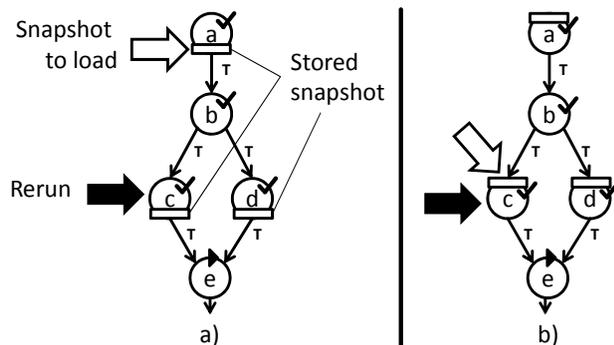


Figure 6. Storing snapshots after variable-modifying activities (a) vs. storing snapshots before variable-modifying activities (b).

Snapshots are stored with every activity that changes variables. If snapshots are saved after variable-changing activities, the workflow graph must always be traversed to

find the correct snapshot for an “iterate” or “re-execute” operation. In Figure 6a, the workflow ought to be rerun from activity c . But the nearest previous snapshot of c belongs to activity a . Another approach is to store the snapshots before variable-changing activities. In Figure 6b, the rerun starts from the variable-modifying activity c and the snapshot of c can be loaded without a need to traverse the workflow graph. Storing snapshots before the execution of variable-changing activities renders the finding of a snapshot for the start activity of a rerun more efficient.

However, if the start activity of a rerun is a non-variable-changing activity, an algorithm is needed to find the nearest preceding snapshot. The simplest case is a sequence of activities. The snapshot to be considered belongs to the nearest preceding variable-modifying activity. In Figure 7a, the rerun is started from activity b , the corresponding snapshot to load belongs to b 's predecessor, activity a . A more complex case arises when there are competing snapshots in a parallel branching. In Figure 7b, the rerun is started from activity e . There are two nearest preceding snapshots located in the branching just before e , one for activity c and one for activity d . This conflict can be solved by the user by manually selecting the snapshot to load. An automatic solution would be to take the snapshot with the youngest timestamp.

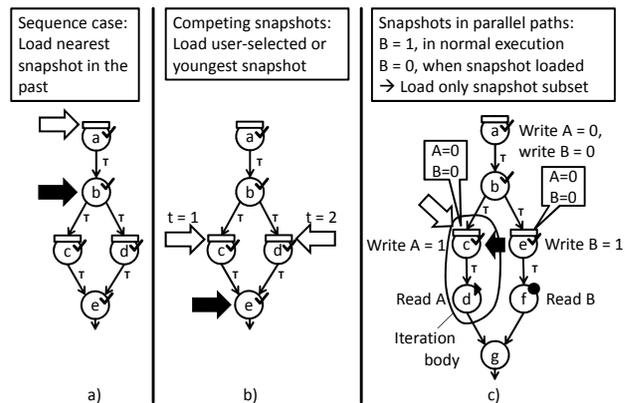


Figure 7. Iteration with non-snapshot activities as start activity with unique (a) and competing (b) snapshots. Iterations in parallel paths can cause the problem of lost updates (c).

In parallel paths, the problem of lost updates might occur: loading a snapshot in one path might overwrite the result of a write operation in a parallel path. A simple example is given in Figure 7c. The two global variables A and B are both initialized with 0 by activity a . In the c/d -branch, activity c increases A by 1 and activity d reads A ; in the e/f -branch activity e increases B by 1 and activity f reads B . The snapshots of c and e have stored the initial values of A and B . Imagine that c and e are already executed and hence variables A and B have the value 1. Now, the user wants to rerun branch c/d starting with c . The snapshot of c is loaded. Both variables get the value 0, which is correct for A but means a lost update for B . The problem cannot be solved by loading another snapshot in the near environment of activity c (the snapshots of a , c and e have the same

content). It must therefore be possible to load only a subset of variables stored in a snapshot. If this is done manually, the user must be able to gain insight into the content of snapshots and to determine the variables to load. An automatic solution is also feasible: all variables that are written in the iteration body can be selected out of the snapshot. In the example, the iteration body consists of activities c and d. The only write operation in the iteration body targets variable A. Hence, variable A can automatically be selected from the snapshot stored before c.

Due to the rerun of activities (manually or in loops) there can be several snapshots for each variable-modifying activity—one snapshot for each execution of the activity. These multiple snapshots are called *snapshot instances*. The user must be given the means to select the particular snapshot instance to be used for the rerun; recommendations may facilitate correctness. In Figure 8, activities c and d of the sample workflow in Figure 7c are iterated multiple times. This leads to a chain of executions of activities c and d. The current value of variable A was taken for each rerun. There are now three snapshot instances for activity c with different values for variable A. Imagine the user wants to iterate again from c. Besides the start activity, he has to select the variables that should be loaded out of the snapshot (variable A) and the concrete snapshot instance. The snapshot instance is identifiable via the execution number of the corresponding activity. For example, the snapshot with A = 100 belongs to the 1st execution of activity c; A = 101 belongs to the 2nd execution of c; and so on. The selected variables of a snapshot instance are re-initialized according to the values stored in the snapshot.

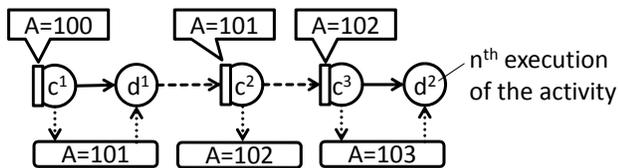


Figure 8. Multiple snapshot instances can exist for one activity.

Now, a function to load snapshot instances can be defined. This function is used by the “iterate” and “re-execute” operations to deliver the correct input for the next enforced run of the corresponding workflow logic.

Definition “Load Snapshot”. The “load snapshot” function loads variable instances for a process instance and a subset of variables. The signature of the function is defined as follows:

$$\text{loadSnapshot}: A \times P_{\text{all}} \times \mathbb{N} \times \wp(V) \mapsto \wp(V^I). \quad (21)$$

Let $a \in A$ be the activity the snapshot belongs to, $p_g \in P_{\text{all}}$ an instance of process model g , $e \in \mathbb{N}$ the execution number for activity a that identifies the snapshot instance, and $V' \in \wp(V)$ the selected variables to load from the snapshot. Then $\text{loadSnapshot}(a, p_g, e, V') = \{v_1, \dots, v_n\}$,

$v_1, \dots, v_n \in V^I \Leftrightarrow \forall k \in \{1, \dots, n\}: \text{model}(v_k) \in V'$, i.e., variable instances are loaded only for the given variables.

V. REPETITION IN COMPLEX WORKFLOW GRAPHS

In activity sequences, the wavefront consists only of a single element and there are no concurrent and hence no join nodes, which does not pose any complications for iteration and re-execution. This section shows the application of the two ad hoc rerun operations in complex workflow graphs with parallel and alternative branches. The most important issue to solve is to guarantee a correct behavior at join nodes when iterating or re-executing them. In common workflow languages, a join node is activated not until all incoming links are evaluated and the join condition is evaluated to `true`. Consider the example in Figure 9. The join node d has three predecessors, a, b and c. Only if the links a-d, b-d and c-d are followed, and the join condition of d is `true`, can d be scheduled (Figure 9a-c). This join behavior prevents (1) missing link values in the join condition of join nodes, and (2) race conditions, i.e., undesired multiple executions of join nodes or ambiguous behavior.

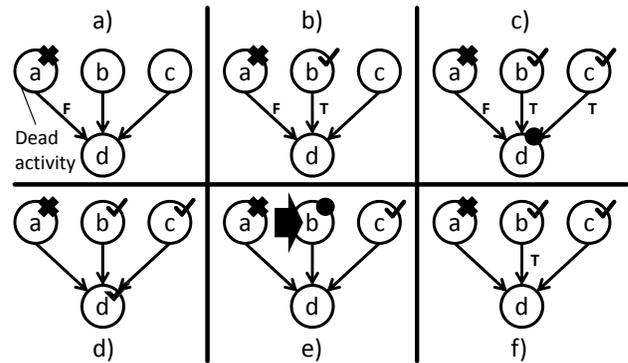


Figure 9. Join behavior of the meta-model. A join activity cannot be executed unless all incoming links are evaluated (a-c). Typically, link states are not stored by the engine (d), which makes a rerun in parallel or alternative branches impossible (e and f).

After the evaluation of the join condition, the values of incoming links of the corresponding activity are not needed anymore. Thus, link values are usually not stored beyond the context of the join node (Figure 9d). This is the typical way of dealing with links in conventional workflow engines. In Petri net-based workflows [19] and BPMN [14], link values are tokens that get consumed by the transition of a join node or a gateway, respectively. In BPEL [15], the link values are bound to boolean variables that are visible only in the context of the target activity instance. That means these variables are destroyed after the execution of the target activity and hence the old link states are lost. As a consequence, a join activity in the iteration body of an “iterate” or “re-execute” operation can lead to a deadlock. In Figure 9e, the iteration body consists of activity b and the join activity d. After the execution of b activity d would never be executed because of the missing states of links a-d and c-d (Figure 9f). Storing the set of evaluated links L^E in

the context of process instances (see Definitions “Evaluated Links” and “Process Instance”) helps solving this problem. The following different use cases show the application of the concept in different situations.

A. Start activity is in a completed AND-branching

The first case discussed is the one in which the start activity for the rerun is located in a parallel, already completed branch. That means the join activity that closes the branching is at least scheduled. Figure 10 shows an example of this case. The parallel branching of activities b to g is completed. The user requests an iteration or re-execution from activity c (Figure 10a). The path beginning with c is followed forward to the wavefront (Figure 10b). All links on this path are reset; all activities in the wavefront reachable from c are terminated (activity h). In case the user wants to re-execute the workflow logic, all completed activities on the path from the start activity c to the wavefront that have assigned a compensation activity (here: c, e and g) are compensated in reverse execution order. Note that the other path of the considered parallel branch, containing activities d and f, and the branch i to m remain unchanged. Finally, start activity c is scheduled (Figure 10c). The wavefront now consists of the activities l and c and the link f-g. In the course of the further execution of the process instance the join activity g can run normally since the state of link f-g is stored in L^E .

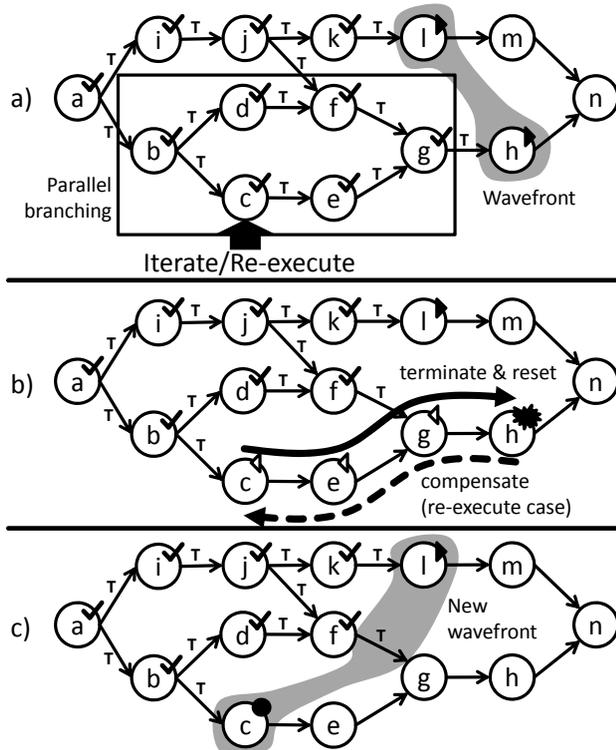


Figure 10. Rerun in an AND-branching.

B. Start activity is in a completed XOR-branching

The rerun in an already completed XOR-branching is very similar to the AND-branching case. In the meta-model, an XOR-branching is achieved with the help of mutual excluding transition conditions of outgoing links of split nodes. In Figure 11a, the link b-c was evaluated to true, whereas b-d is false. Hence, activities b through g implement an alternative branch. The path containing the link b-d, the activities d and f and the link f-g is dead. The join behavior in the meta-model requires all links to be evaluated before a join node can be executed. That is the reason why an algorithm for dead path elimination (DPE) is used to set all links in a dead path to false [5]. In the example, this holds for the links d-f and f-g. Activities on a dead path are not executed; their state is simply set to dead (activities d and f). In this scenario, the user wants to rerun the workflow from activity c which is located in the completed path of the XOR-branching. Due to the DPE and the set of all evaluated links L^E , this case can now be addressed exactly the same way as the ad hoc rerun in completed AND-branches.

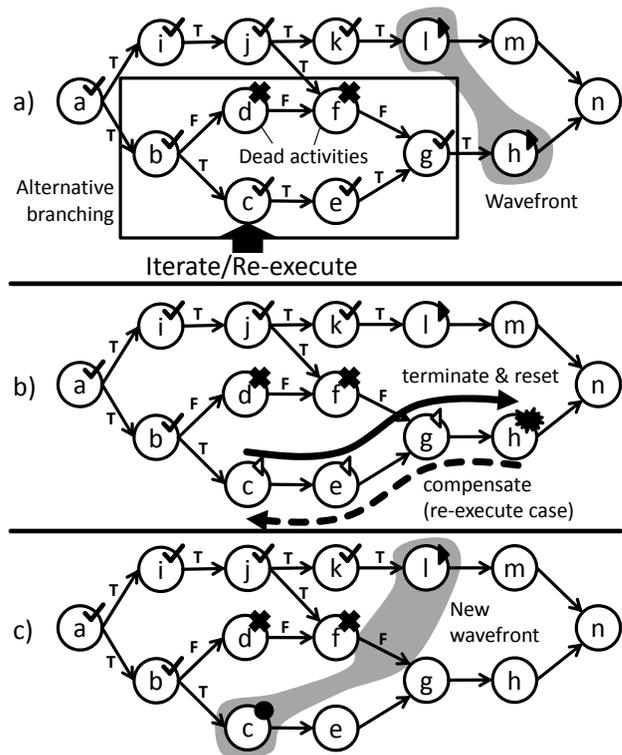


Figure 11. Rerun in an XOR-branching.

Note that there are cases where a branching in a process model can be an AND in some process instances and an XOR in other instances. It depends on the selected transition conditions and the process context (e.g. variable values) if one, all or a particular number of branches is followed during workflow execution. In BPMN, this behavior can be modeled with an inclusive gateway [14]. However, such

cases are covered by the concept because links in dead branches are evaluated (to `false`) in the course of the DPE.

C. Start activity is before a branching

In this case, the start activity is located before a branching, i.e., the iteration body contains branching activities. In Figure 12a, the user reruns the workflow with b as start activity. The two outgoing links of b show that it is a branching activity. In order to address this case, all paths beginning with b are followed to the wavefront (Figure 12b). All visited links are deleted from L^E (b-d, d-f, f-g, b-c and c-e) and all visited scheduled or running activities are terminated (e). In the re-execute case, the reachable completed activities that can be compensated (c and d) are compensated in reverse execution order. After that, the start activity is scheduled (Figure 12c) and the workflow can be resumed.

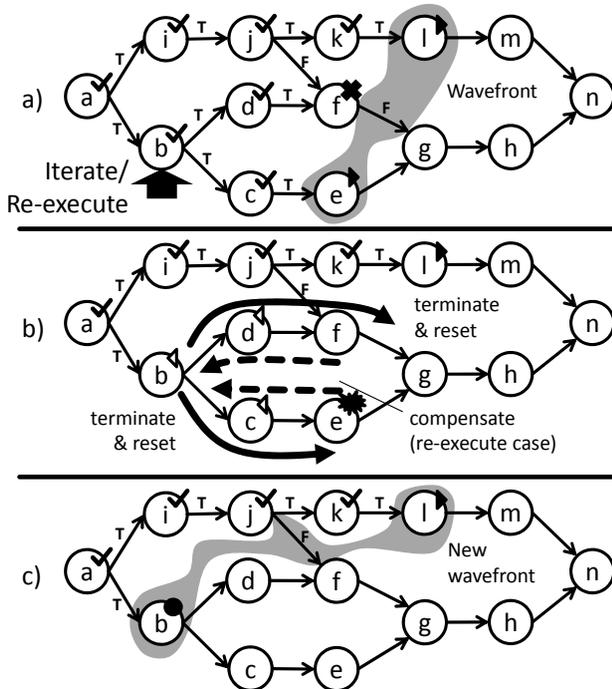


Figure 12. Rerun when the start activity is located before a branching.

Figure 13a shows a more complex scenario. Note that the difference to the previous workflows is that link j-f was deleted and link d-k added. The user wants to rerun the workflow beginning with activity b. The iteration body thus contains two branching (b and d) and two join activities (g and k). All links on the path from b to the wavefront are reset and all scheduled/running activities are terminated (Figure 13b). It is sufficient to visit the activities and links only once with the algorithm, like activities g and h and link g-h. In the re-execution case, the completed activities on the considered path are compensated in reverse execution order (g, f, e, d, and c). The start activity (b) is then scheduled and the rerun operation is complete (Figure 13c).

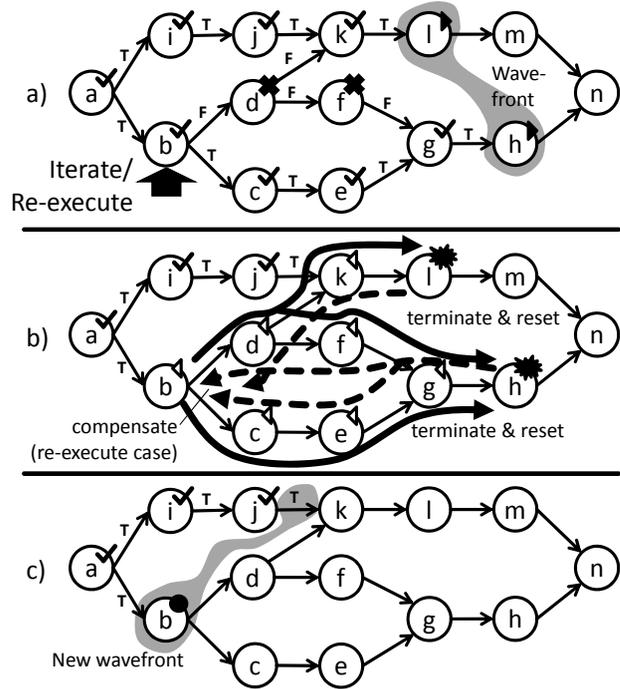


Figure 13. Complex rerun scenario with several branches.

D. Repetition in Dead Paths

As shown above the concept to enforce the rerun of workflow logic can be applied in sequences of activities, in parallel and alternative branches, and in complex graphs. An interesting research question is (1) whether the start activity of a rerun can be located in a dead path and (2) whether such an operation would be meaningful. Note that there is a difference between a dead path and a path that is not (yet) executed. A dead path belongs to the past of the workflow instance while a not executed path is the future of the workflow instance. That means the latter is a jump to the future, that can be realized by a “skip” operation, which, however, is not part of this work.

The precondition of the “iterate” and “re-execute” operation is that the state of the start activity is scheduled, executing, completed, faulted or terminated (see Section III.A). The iteration/re-execution in dead paths is thus not allowed. However, if this precondition was neglected, repeating activities in a dead path would technically be possible with the presented concept. As an example consider Figure 14a. The user requests the repetition of activity f, which is located in a dead path. As usual, the path from f to the wavefront is followed, links are reset (f-g and g-h), running activities terminated (h) and completed activities compensated (g, in case of a re-execute), as shown in Figure 14b. Then, the start activity f is scheduled and can be executed when the workflow is resumed. Although conceptually feasible, the operation has several problems. The result is obviously an unrealistic execution history. Activity f gets executed although its predecessors d, i and j were not enacted (Figure 14c). Further, the operation is not

really an iteration or a re-execution because at least the start activity was not executed before the operation. Hence, it is not a repeated execution of activities but rather an ad hoc change of a process instance that enforces the execution of a dead path. That is why the above-mentioned precondition prevents a repetition of activities in dead paths.

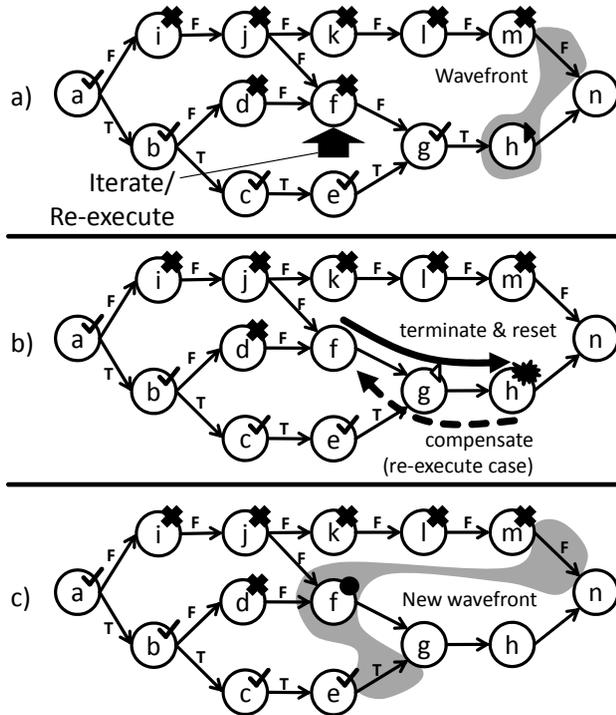


Figure 14. Rerun in a dead path.

However, although it is not recommend using the “iterate/re-execute” operation in dead paths, a workflow system implementing the approach should provide as much flexibility as possible for scientists or other users. The user must decide whether the operation helps to achieve the desired goals. With the help of the precondition the workflow system is able to detect that the user is about to conduct an ad hoc rerun in a dead path. The user should then be requested if he really wants to apply the operation in a dead path and if so, the system conducts the operation as shown in this section.

VI. IMPLICATIONS ON THE EXECUTION CORRECTNESS

Workflows consist of different activity types, e.g., for sending/receiving messages, loops, or variable assignment. The enforced repetition of workflow logic has to account for different activity types, especially those that interact with external entities such as clients, humans, or services/programs. The main problem is that the repetitions are not reflected in the workflow logic because they are an ad hoc user operation. Hence, the aforementioned external entities do not know a priori the exact behavior of the workflow. An uncoordinated rerun of workflow logic can lead to multiple invocations of services, multiple identical

work items in the work list for human users or an infinite waiting for messages because the communication partner does not know that a message must be sent again.

A. Message-receiving Activities

If a message receiving activity is repeated, it would wait infinitely for the message because it was already consumed. Three cases can be distinguished to solve this problem. Firstly, the original message sent by the partner in a former iteration is taken as incoming message for the next run of the activity. However, if the activity was iterated several times, there may be different versions of the incoming message. The user then has to select the desired message.

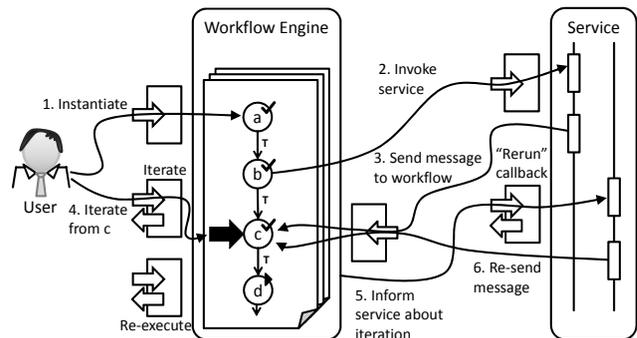


Figure 15. Repetition of a message-receiving activity. The communication partner is informed about the rerun of the activity over a special “rerun callback” operation. The partner then can re-send the message or send an adapted (i.e., updated) message to the engine.

Secondly, the message sending partner re-sends the message or sends an adapted message. The partner needs to be informed about the repetition of the activity. A simple solution is that partners provide a special “rerun callback” that can be used by the workflow engine to propagate the iteration or re-execution. An architecture for this concept is shown in Figure 15. The workflow engine provides the “iterate” and “re-execute” operations. The considered workflow that is deployed on the engine implements two operations, one for its instantiation and one to receive a message from a service. The course of action in this setting is as follows. The user invokes (i.e., instantiates) the workflow (1). The workflow calls a matching service in an asynchronous manner (2) and provides a callback for the response (activity c). The invoked service creates the response message and sends it to the engine (3). The engine correlates the message to the particular workflow instance (e.g., via the instance id or some other information that uniquely identifies the workflow instance). Now the user decides to iterate the workflow logic with activity c as start activity and invokes the corresponding “iterate” operation (4). The workflow then waits again for the message of the service. The engine informs the partner about the iteration that took place in the workflow instance. This is done by invoking the special “rerun callback” provided by the partner (5) or a mechanism in the service infrastructure performing the same functionality. The engine’s message contains at least the following information: the original message of the

partner (in case the partner did not persistently store the message), the engine's address, and correlation information to identify the workflow instance. The partner then decides whether to re-send the message or to send an adapted one (6). Sending an adapted message is useful if the information distributed by the partner has to be updated (e.g., sensor data). The engine has to find the partners to be considered in order to invoke their "rerun callback". It first searches for all message-receiving activities in the iteration body. Then, it determines the addresses of the related partners. The addresses can be found in the "ReplyTo" header field of a message received in a former run of the workflow logic (if WS-Addressing [20] is used). Or it is taken from a message that was sent to a partner by a message sending activity in the same workflow instance. This scenario has the disadvantage that it has many implications on the partners' services and/or infrastructure and would be difficult to enable in a standard manner.

Thirdly, message-receiving activities are usually related to message-sending activities. The workflow system or the user pays attention that if a message-receiving activity is iterated, its corresponding message-sending activity is iterated, too. The reason is that an incoming message is often the response to a message sent to a partner. Hence, repeating the invocation of the partner will make the partner send the message again to the workflow engine (or an adapted message with updated content).

A workflow system that implements the ad hoc rerun should support all three cases. It depends on the implemented message exchange pattern, on the concrete function realized by the partner and on whether the partner is stateful or stateless to select (possibly with user-support) the adequate mechanism for the repetition of message-receiving activities.

B. Message-sending Activities

The repetition of message sending activities is straight forward for idempotent services. Non-idempotent services should be compensated prior to a repeated invocation, as is done in the re-execution operation. If the iteration operation repeats the execution of non-idempotent services, then the user is responsible for the effect of the operation.

C. Loops

Iterations within modeled loops can have an unforeseeable impact on the behavior of workflows. The context of workflows might be changed in a way that leads to infinite loops (e.g., because the repetition changes variable values so that a while-condition can never evaluate to `false`). Usually, a workflow system provides operations to change variable values (e.g., in a process repair component). This functionality can be used to resolve infinite loops.

It can also happen that the start activity of an iteration or re-execution is located within an already completed loop. The operation causes the loop to run again. In its first iteration the loop body begins with the start activity of the ad hoc rerun. From its possible second iteration on the complete loop body is executed. The user must be able to select the particular iteration of the loop that should be repeated. This can be done with the help of a variable snapshot loaded prior

to the rerun because the former variable values represent the iteration of a loop (e.g., via a counter variable).

D. Workflow Languages with Block Structures

In practice, a workflow is not a simple graph consisting of nodes and edges. There are often also hierarchically nested elements. In BPMN, there is the concept of sub-processes that are containers for arbitrary workflow logic [14]. BPEL offers structured activities (e.g., the `sequence`, `flow`, or `while` activity) that can contain other activities for a simplified modeling of complex workflows [15]. The rerun of activities in hierarchical structures has to account for parent-child-relationships of activities.

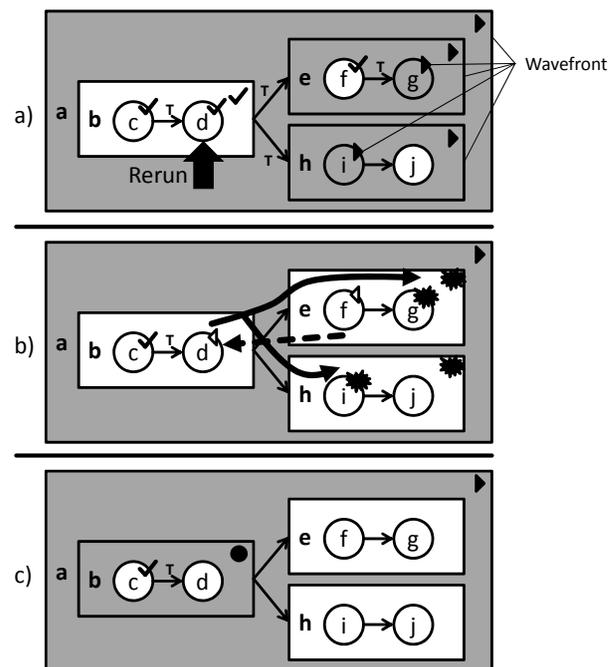


Figure 16. Rerun in workflows with block-structures

Figure 16 illustrates an example for the rerun in hierarchically nested activities. The process model contains a sequence of activities c and d followed by a parallel branching of f/g and i/j (Figure 16a). Because of the nesting, the wavefront of the considered process instance (the shaded activities) is stretched across the complete process. All parent activities with at least one active child are also active. Hence, the termination and resetting of the path from the start activity to the wavefront is more complex. In the example, d is the start activity of the rerun (Figure 16a). The atomic activities d, f, g and i as well as the parent activities b, e and h have to be terminated or reset (Figure 16b). Moreover, the start activity cannot be simply scheduled. It must be checked whether the corresponding parent activity is active. If so, the start activity can be scheduled. If not, the parent must be scheduled instead of the start activity itself (and possibly the parent of the parent, etc.). It is important to make sure that, although the parent activity is scheduled, only a subset of its child activities will be executed. In Figure

16c, the parent activity b of the start activity d is scheduled. But it must be ensured that activity c is not executed again. Realizing this behavior in a workflow engine is highly implementation-specific.

E. Impact on Scopes

In modern workflow languages such as BPMN or BPEL, the concept of scopes is used to denote containers for activities, data objects and correlation keys; they span transaction boundaries and specify fault handling logic as well as logic to handle incoming events. At the beginning of their execution, scopes initialize their context. That means fault handlers and event handlers are installed and local variables are instantiated.

If a rerun is conducted with the start activity being located in an already completed scope, this scope has to be scheduled because of the parent-child relationship discussed before. The scope's context has to be initialized again. In case of a re-execution, the scope's effects have to be undone before the workflow can be resumed. Invoking the scope's compensation handler undoes the work of the complete scope. This is the desired behavior only when the start activity is the first activity in the scope. Otherwise the specific compensation handler of the scope must not be executed, but rather only the compensation handlers of the activities following the identified start activity in the reverse execution order.

The repetition of activities also has an impact on fault and compensation handlers attached to scopes. Fault and compensation handlers can be used to undo already completed work. If logic is rerun within these handlers, it must be ensured that the corresponding scopes are not compensated multiple times.

VII. USER INTERACTION WITH THE WORKFLOW SYSTEM

A workflow system that implements the ad hoc rerun of workflow logic must provide a monitoring tool that allows users to continuously follow the execution state of process instances (see Figure 17(1)). The user interacts with the system as follows. If the user detects a faulty or unintended situation, he can suspend the workflow (2) and manually trigger an iteration/re-execution (3).

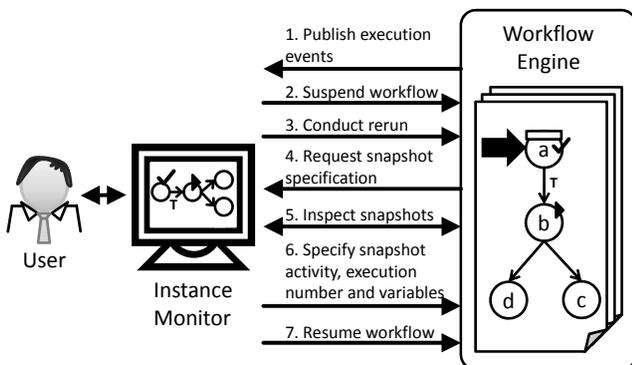


Figure 17. User interaction with a workflow system that implements the concept of the enforced repetition of workflow logic

The workflow system asks him to specify which snapshot instance should be taken for the rerun and which contained variables should be loaded (4). The user can inspect different snapshot instances and the values of their variables in order to determine the desired snapshot instance (5). He specifies the snapshot instance with the corresponding variable-modifying activity, the execution number of the activity, and the subset of variables to be loaded. The process instance state is changed in the engine as described in Section III through V. Finally, the user resumes the workflow instance (7). Note that steps 4 to 6 are omitted if the user conducts an iteration of activities with the current variable values, i.e., without loading a snapshot.

VIII. IMPLEMENTATION

The implementation of the “iterate” and “re-execute” operations is based on the Apache Orchestration Director Engine (ODE) [21] as BPEL engine and on the Eclipse BPEL Designer [22] as GUI for the users of the system.

A. Architecture of the System

Figure 18 shows the high level architecture of the workflow system that implements the ad hoc rerun of workflows. Components with dashed lines are new or extended. The scientist/user interacts with Eclipse and the BPEL Designer plugin in order to model and run workflows. The Execution Control component enables starting of workflows directly in the BPEL Designer. A special dialog requests the user to specify the content for the input message. Deployment of workflows happens transparent for the user. The underlying workflow engine is hidden. Workflow instances can be suspended and resumed. The Instance Monitor visualizes the current execution state of running workflows by coloring activities and links. The scientist can inspect and change values of variables and endpoint references assigned to partner links.

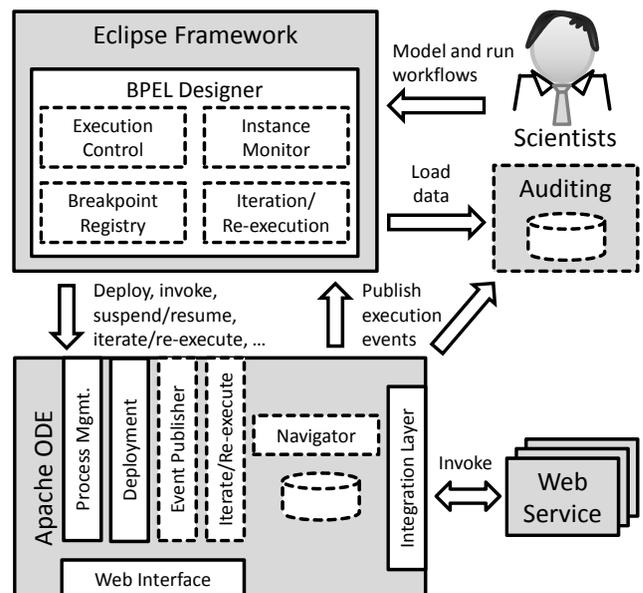


Figure 18. High level architecture of the prototype

In order to suspend workflows at points of particular interest for the user, it is possible to set breakpoints at activities or links in a *Breakpoint Registry*. The *Iteration/Re-execution* component provides the ad hoc rerun operations to the user. A wizard helps the user to find and select the desired snapshot instance to load prior to the rerun. The needed information is fetched from the workflow engine. When a rerun is conducted, the activity states of the instance monitor are refreshed and an iterate/re-execute operation of the engine is invoked.

Apache ODE provides interfaces to deploy and undeploy processes (*Deployment* component) and to access information about process models and instances (*Process Management* component). Web services are invoked over an *Integration Layer*. There is also a *Web Interface* for user, which does not play a role in this work. The Apache ODE was extended with an *Event Publisher* that emits execution events of workflow instances, e.g., activity ready, activity running, or link evaluated. These events are received by the BPEL Designer's Instance Monitor and used to color workflow activities and links. The *Navigator* is the heart of the workflow engine. It traverses the workflow graph and executes activities. An extension of the Navigator and the database is that variable and partner link values are stored as snapshot before the execution of variable changing activities. The new *Iterate/Re-execute* component provides the two rerun operations to clients. The component loads a variable/partner link snapshot according to the input of the user. Then, the execution queue of the navigator is adapted: activity instances that have to be terminated are removed from the execution queue; a new instance of the start activity is scheduled (and possibly new instances of its parent activities), i.e. put to the execution queue.

An *Auditing* component external to the workflow engine stores the published execution events persistently. The BPEL Designer makes use of the Auditing to load the state of a workflow instance into the Instance Monitor. This has the advantage that the engine's execution events are not lost even if Eclipse is shut down during workflow execution.

The following two sections provide more details on the extensions of the BPEL Designer and the Apache ODE.

B. Extensions of the BPEL Designer

The scientist can use the functions of the Execution Control from the extended toolbar menu (Figure 19a). A workflow can be started, suspended, resumed and terminated. If a breakpoint is reached during execution, a skip breakpoint operation releases the breakpoint and the workflow execution proceeds.

In order to implement the Instance Monitor the Eclipse Modeling Framework (EMF) core model for BPEL was extended with a *state* attribute for all activities and links. It holds the state of activities/links based on the execution events of the engine. The state indicates the color of each element (Figure 19b): yellow is running, green is completed, red is faulted, orange means a breakpoint is reached, and grey are dead activities.

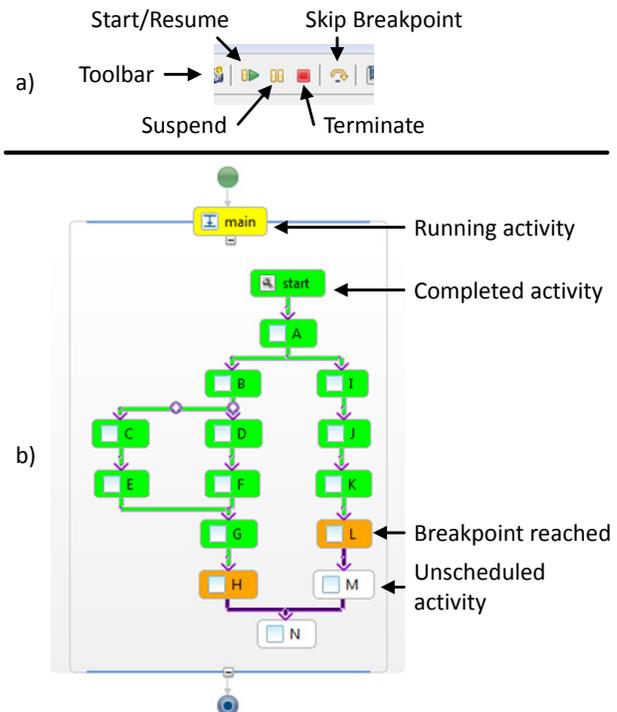


Figure 19. BPEL Designer extension: (a) Execution Control in the toolbar and (b) the Instance Monitor.

When a workflow is suspended, the user can iterate or re-execute workflow logic via the context menu of an activity (Figure 20). The selected activity is then the start activity of the operation (activity B in the figure).

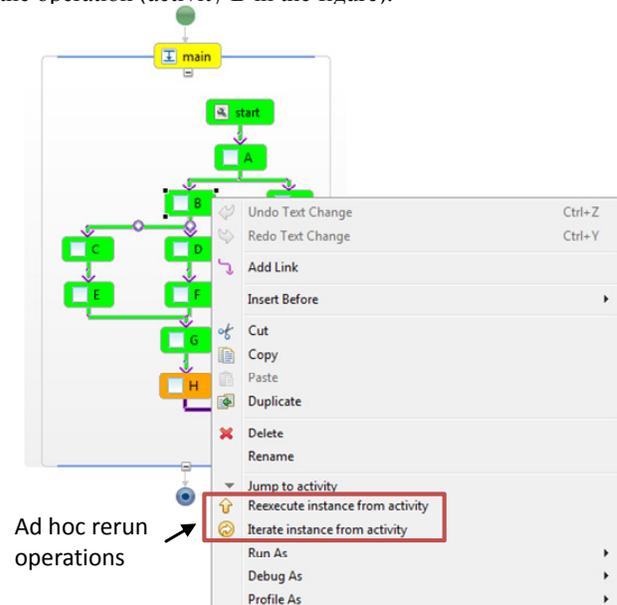


Figure 20. The user can select the iterate/re-execute operations from the context menu of activities.

A wizard opens that guides the user step by step through the snapshot selection process. First, the activity to load the

snapshot for has to be chosen. This can be the start activity or a predecessor thereof. The latter can happen when the start activity is no variable-changing activity and hence does not

possess a snapshot. The wizard shows all snapshot instances for the selected activity.

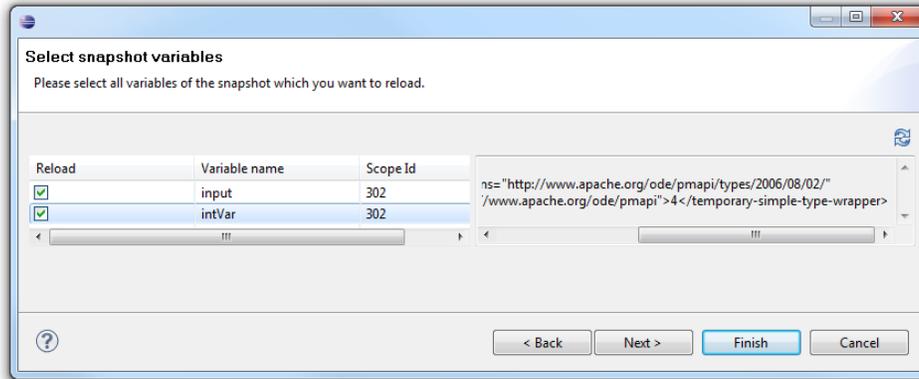


Figure 21. Wizard to select variable snapshots.

The user can have a look at the snapshot content, i.e., at the values of stored variables (Figure 21) and partner links. It is possible to select only a subset of stored variables (Figure 21) and partner links to be loaded in the course of the rerun, which can prevent a lost update of variables in parallel paths. When the snapshot selection is done, the Instance Monitor is refreshed, i.e., the state of all activities in the iteration body is set to inactive. Finally, the iterate/re-execute operation of the Apache ODE is invoked.

C. Extensions of the Apache ODE

The Navigator was extended so that each variable-changing activity persistently stores a snapshot with all visible variables and partner links before its execution. This pertains to receive, pick, invoke and assign activities. Three new tables are created in the database to store the snapshots (Figure 22). The table ODE_SNAPSHOT holds information about snapshot instances: the corresponding process instance, the scope the stored variables and partner links belong to, the creation time, the version, and an XPath expression pointing to the corresponding activity. The table ODE_SNAPSHOT_VARIABLE stores the concrete values of variables that belong to a snapshot. And finally, the table ODE_SNAPSHOT_PARTNERLINKS holds the values of partner links stored in snapshots. A partner link can have up to two values, one EPR for each of the at most two roles. There is another new table, ODE_LINK_INSTANCE, used to save the state of link instances as discussed in Section V.

The Web service interface of Apache ODE was extended with five operations. The *iterate/re-execute* start the ad hoc rerun for a specific workflow instance. Both require the process instance ID, the XPath expression of the start activity, the XPath expression of the activity to load the snapshot for, the snapshot version (i.e., the execution number), and a list of variables and partner links to load. The *getSnapshots* operation delivers all snapshot instances for a given process instance and activity, but without loading the concrete values of the stored activities/partner links; the

getSnapshotPartnerLinks and *getSnapshotVariables* operations are then used to load concrete values out of a snapshot identified via the process instance and snapshot ID. This functionality is distributed on several operations for the sake of smaller messages. It is often sufficient to load just some general information about a snapshot and not all the contained values.

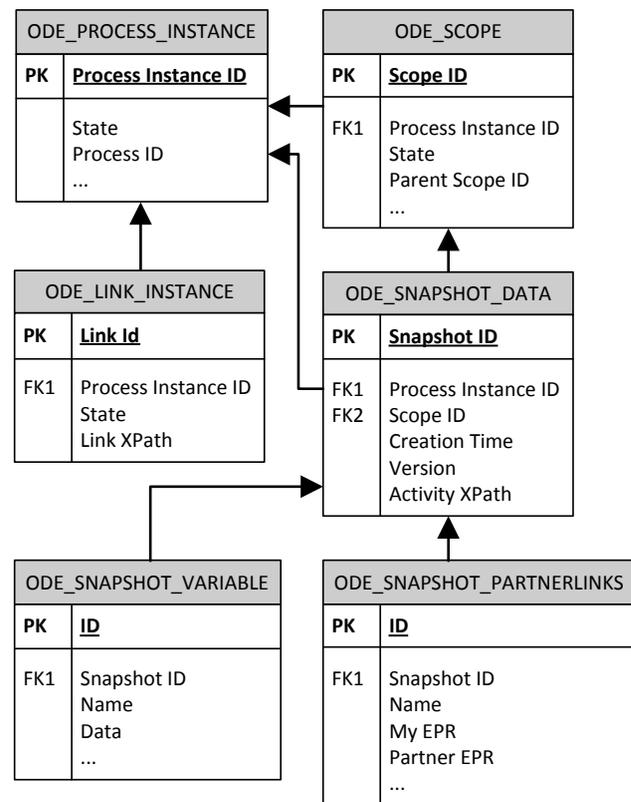


Figure 22. Extension of the database schema to store variable/partner link snapshots and the state of link instances.

The two most critical parts of the ad hoc rerun are (1) to correctly and consistently adapt the content of the execution queue and (2) to adapt the activity instance of the start activity's parent. In Apache ODE, the execution queue is an object that holds a list with all scheduled activity instances, another list with all channels used to send information between activities (e.g. a child activity uses a channel to inform its parent about its completion), and a third list with completed activities. All activity instances and channels that belong to activities in the iteration body have to be removed from these lists.

The modification of the start activity's parent has to be implemented per activity type. It is currently realized for `sequence` and `flow` activities. In a `sequence`, the `sequence` activity instance is scheduled again but only with the start activity and all successor activities as children. All activities preceding the start activity are omitted because they do not belong to the iteration body. In a `flow`, all completed activities of the former iteration have to be marked as *not completed* and are scheduled again. Their activity guards make sure that the activities are executed not until their join conditions can be evaluated. Only the start activity of the rerun is executed without evaluation of its join condition.

IX. RELATED WORK

The term "iteration of activities" is mentioned in [5] as one of the change operations that can be performed in a workflow; no details are available about how iteration should be performed. In ADEPT, it is possible to perform manual ad hoc backward jumps that are similar to the rerun operations in this paper, as claimed in [16]. The target activity of the jump is executed again. The previous execution state is restored based on the execution and data element history. While in [16] it is said *that* an operation for ad hoc backward jumps exists, no details such as algorithms, applications on workflow languages with hierarchically nested elements, or impact of different activity types are provided as is done in this work. In the scientific workflow system e-BioFlow, scientists can re-execute manually selected tasks with the help of an ad hoc workflow editor [6]. The set of activities that should be (re-)executed must be marked explicitly. No other activities are (re)executed; no distinction is made between iteration and re-execution operations. Following the approach in this paper, the user only has to provide the start activity for the rerun and the successor activities are then executed as prescribed by the workflow model.

Repetition of workflow logic can be achieved language-based with certain modeling constructs. A general concept to retry and rerun transaction scopes in case of an error is shown in [23] for the case of business transactions. Eberle et al. [10] apply this concept to BPEL scopes. In BPMN [14] this behavior can be modeled with sub-processes, error triggers and links. In IBM MQSeries Workflow a Flow Definition Language (FDL) activity is restarted if its exit condition evaluates to `false`. ADOME [24] can rerun special repeatable activities if an error occurs during activity execution; the approach is applicable only for single

activities, not for groups of activities. In Apache ODE, an extension of BPEL's `invoke` activity enables retrying a service invocation if a failure happens [25]. These approaches have special modeling constructs in common to realize the repetition. In these cases, the rerun is pre-modeled at design time. In contrast to these approaches, the solution in this paper aims at repeating a workflow starting from an arbitrary, not previously specified point.

Iterations can also be realized by configuring workflow models with deployment information. Invoke activities in the Oracle BPEL Process Manager [11] can be configured with an external file so that service invocations are retried if a specified error occurs. The concept to retry activities until they succeed is also subject of [26] and also in [27] where the service selected for the retry is identified using a semantic description of selection criteria. The scientific workflow system Taverna [28] allows specifying alternate services that are taken if an activity for a service invocation fails. In contrast to these and other available similar approaches, this paper advocates a solution where the rerun can be started spontaneously without a pre-configuration of workflows from an arbitrary point.

The scientific workflow system Pegasus can automatically re-schedule a part of a workflow if an error occurs [12]. Successfully completed tasks are not retried. The Askalon workflow system provides a checkpointing-like functionality to handle runtime faults [29]. Kepler's Smart Rerun Manager can be used to re-execute complete workflows [30]. Tasks that produce data that already exists are omitted. The main difference of these approaches to this paper is that the ad hoc rerun allows selecting the starting point of the iteration (manually) and hence this functionality can be used for different purposes, e.g. explorative workflow development, steering of the convergence of scientific results, or fault handling.

Checkpointing in workflow management is a technique to store the complete workflow state at specific execution points geared towards transactions spheres [31]. If a failure happens, these checkpoints can be used to rollback a workflow, i.e., load its former state, and run a part of the workflow again. Assurance points (AP) [32] are a similar concept that store data at critical points in a workflow. APs are user defined at modeling time and enable backward recovery of a complete process, retry of a workflow part, and forward recovery. Compared to the approach in this paper, checkpoints and assurance points cannot be used to rerun a workflow part starting from an *arbitrary* activity chosen at runtime. Apart from this, the retry functionality of APs can be compared to the re-execute operation in this work because already completed work from the current wavefront to the AP is compensated. In [33], an aspect-oriented approach for dynamic checkpointing in workflows is introduced. It allows selecting and changing checkpoint positions at workflow runtime in order to transfer running workflows from one to another workflow engine instance. The approach can be used to rerun activities of a workflow in an ad hoc manner. In contrast to the approach in this paper, the rerun would require an additional step: the selection of an adequate checkpoint in the future of a workflow instance that will be

the target of a rerun later on. Thus, the scientist must prepare a rerun before the execution of the workflow part, which is more restrictive than the ad hoc rerun proposed in this work.

In [34], the authors present and describe several types of flexibility in process-aware information systems. The option “Undo task A” in the flexibility type “Flexibility by deviation” is similar to the iterate/re-execute operation in this work. The control is moved back just before the execution of a task (= iteration); in some cases, it is meaningful to compensate already completed work (= re-execution). No further details are provided about data issues, how race conditions are avoided, how parallel/alternative/dead paths are dealt with, or how block-structures influence the approach as it is done in this work.

X. CONCLUSION AND FUTURE WORK

This paper dealt with the formal description of two operations to enforce the rerun of workflow logic during workflow execution: the *iterate* operation reruns activities starting from a manually selected activity; the *re-execute* operation undoes completed work in the iteration body before rerunning activities. The distinctive features of the approach are that the repetition does not have to be modeled or configured previously and that arbitrary activities can be used as starting point for the rerun. It was shown that the approach can be applied in sequences of activities, parallel and alternative branches as well as in more complex scenarios that include the repetition of join activities. Furthermore, an adoption of the operations in dead paths has been investigated. An ad hoc rerun in dead paths is not recommended because it is literally no rerun of activities. But it should be up to the user to decide about the meaning of such an operation. One of the main issues when repeating activities is the question which data to take as input for the next run. This issue is addressed with the help of data snapshots that are stored before each variable-modifying activity and that are loaded in the course of the rerun.

Real world processes depend on external communication partners, services or clients. An operation for the repetition of activities has to account for dependencies on messages from partners and on the impact of repeatedly delivered messages on services invoked by the workflow. There are three ways to deal with the repetition of message-receiving activities: reuse a message received in a former iteration, inform the communication partner about the ad hoc rerun and the partner re-sends the message, and repeat a message-receiving activity together with its corresponding preceding message-sending activity. Furthermore, it was shown how users interact with such a flexible workflow system. A workflow instance monitor that shows the workflow progress in real-time and that allows an immediate intervention of the user is of utmost importance in this setting. The concepts presented in this paper are based on an abstract meta-model and thus can be applied to existing or future workflow engines and languages. It was shown how the ad hoc rerun works in languages with concepts for block-based modeling and scopes, such as BPEL or BPMN. The implementation of the iterate and re-execute operations for BPEL in the Eclipse

BPEL Designer and Apache ODE evaluate the formal concepts presented in this paper and proof their feasibility.

The enforced repetition of workflow logic is a step towards the goal to enable an explorative workflow development, especially in the field of scientific workflows.

In future, we will also work on an ad hoc “skip” operation that allows omitting activities, e.g., if the result of the respective activities is already present.

ACKNOWLEDGMENT

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

REFERENCES

- [1] M. Sonntag and D. Karastoyanova, “Enforcing the Repeated Execution of Logic in Workflows,” Proc. of the 1st International Conference on Business Intelligence and Technology (BUSTECH 2011), 2011.
- [2] W. M. P. van der Aalst, T. Basten, H. Verbeek, P. Verkoulen, and M. Voorhoeve, “Adaptive workflow: on the interplay between flexibility and support,” Proc. of the 1st Conference on Enterprise Information Systems (ICEIS), 1999, pp. 353–360.
- [3] M. Reichert and P. Dadam, “ADEPT_{flex}—Supporting dynamic changes of workflows without losing control,” Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, vol. 10(2), 1998, pp. 93–129.
- [4] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, “Workflow evolution,” Journal of Data and Knowledge Engineering, Elsevier, vol. 24(3), 1998, pp. 211–238.
- [5] F. Leymann and D. Roller, “Production workflow—Concepts and techniques,” Prentice Hall, 2000.
- [6] I. Wassink, M. Ooms, and P. van der Vet, “Designing workflows on the fly using e-BioFlow,” Proc. of the International Conference on Service Oriented Computing (ICSOC), 2009.
- [7] R. Barga and D. B. Gannon, “Scientific vs. business workflows,” in: I. Taylor, E. Deelman, D. B. Gannon, and M. Shields (Eds.), “Workflows for e-Science—Scientific workflows for grids,” Springer, 2007, pp. 9–18.
- [8] G. Vossen and M. Weske, “The WASA approach to workflow management for scientific applications,” Workflow Management Systems and Interoperability, NATO ASI Series F: Computer and System Sciences, vol. 164, Springer, 1998, pp. 145–164.
- [9] M. Sonntag and D. Karastoyanova, “Next generation interactive scientific experimenting based on the workflow technology,” Proc. of the 21st IASTED International Conference on Modelling and Simulation (MS), 2010.
- [10] H. Eberle, O. Kopp, F. Leymann, and T. Unger, “Retry scopes to enable robust workflow execution in pervasive environments,” Proc. of the 2nd Workshop on Monitoring, Adaptation and Beyond (MONA+), 2009.
- [11] Oracle BPEL Process Manager, <http://www.oracle.com/us/products/middleware/application-server/bpel-home-066588.html>
- [12] E. Deelman, G. Mehta, G. Singh, M.-H. Su, and K. Vahi, “Pegasus: Mapping large-scale workflows to distributed resources,” In: I. Taylor, E. Deelman, D. B. Gannon, and M. Shields (Eds.), “Workflows for e-Science—Scientific workflows for grids,” Springer, 2007, pp. 376–394.

- [13] H. Garcia-Molina and K. Salem, "Sagas," Proc. of the ACM Sigmod International Conference on Management of Data, pp. 249–259, 1987, doi:10.1145/38713.38742.
- [14] Object Management Group (OMG), "Business Process Modeling Notation (BPMN) Version 1.2," OMG Specification, 2009.
- [15] OASIS, "Web Services Business Process Execution Language (BPEL) Version 2.0," OASIS Standard, 2007, <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>
- [16] M. Reichert, P. Dadam, and T. Bauer, "Dealing with forward and backward jumps in workflow management systems," International Journal of Software and Systems Modeling (SOSYM), vol. 2(1), 2003, pp. 37–58.
- [17] R. Khalaf, "Supporting business process fragmentation while maintaining operational semantics: a BPEL perspective," Doctoral Thesis, ISBN: 978-3-86624-344-6, 2008.
- [18] Workflow Management Coalition, "Audit Data Specification, Version 1.1," WfMC Specification, 1998.
- [19] W. M. P. van der Aalst, "The application of Petri nets to workflow management," Journal of Circuits, Systems and Computers, vol. 8(1), 1998, pp. 21–66.
- [20] World Wide Web Consortium (W3C), "Web Services Addressing 1.0 – Core," W3C Recommendation, 2006, <http://www.w3.org/TR/ws-addr-core/>
- [21] Apache Software Foundation, "Apache Orchestration Director Engine (ODE)," <http://ode.apache.org/>
- [22] Eclipse BPEL Project, "Eclipse BPEL Designer," <http://www.eclipse.org/bpel>
- [23] F. Leymann, "Supporting business transactions via partial backward recovery in workflow management systems," Proc. of the Conference on Database Systems for Business, Technology and Web (BTW), Springer, 1995.
- [24] D. Chiu, Q. Li, and K. Karlapalem, "A meta modeling approach to workflow management systems supporting exception handling," Journal of Information Systems, Elsevier, vol. 24(2), 1999, pp. 159–184.
- [25] Apache Software Foundation, "Failure and Recovery in Apache ODE," <http://ode.apache.org/activity-failure-and-recovery.html>
- [26] P. Greenfield, A. Fekete, J. Jang, and D. Kuo, "Compensation is not enough," Proc. of the 7th International Enterprise Distributed Object Computing Conference (EDOC), 2003.
- [27] D. Karastoyanova, F. Leymann, and A. Buchmann, "An approach to parameterizing Web service flows," Proc. of the 3rd International Conference on Service Oriented Computing (ICSOC), 2005, pp. 533–538, doi:10.1007/11596141_45.
- [28] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services," Journal of Nucleic Acids Research, vol. 34, Web Server issue, 2006, pp. 729–732, doi:10.1093/nar/gkl320.
- [29] E. Deelman, D. B. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities," International Journal of Future Generation Computer Systems, Elsevier Science Publishers, vol. 25(5), 2009, pp. 528–540.
- [30] I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance collection support in the Kepler scientific workflow system," International Provenance and Annotation Workshop (IPAW), Springer, LNCS, vol. 4145, 2006, pp. 118–132.
- [31] Z. Luo, "Checkpointing for workflow recovery," Proc. of the 38th ACM Southeast Regional Conference, 2000, pp. 79–80, doi:10.1145/1127716.1127735.
- [32] S. Urban, L. Gao, R. Shrestha, and A. Courter, "Achieving recovery in service composition with assurance points and integration rules (short paper)," Proc. of the OTM Conferences (1), 2010, pp. 428–437.
- [33] S. Marzouk, A. J. Maâlej, and M. Jmaiel, "Aspect-oriented checkpointing approach of composed Web services," Proc. of the 1st Workshop on Engineering SOA and the Web (ESW), Springer, LNCS, vol. 6385, 2010, pp. 301–312.
- [34] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. M. P. van der Aalst, "Process flexibility: a survey of contemporary approaches," Proc. of the 4th International Workshop CIAO! and the 4th International Workshop EOMAS, Springer, LNBIP, vol. 10, 2008, pp. 16–30.

All links were last checked on June 26, 2012.

Modeling Disjunctive Context in Access Control

Narhimene Boustia, Saad Dahlab
University of Blida, Algeria
nboustia@gmail.com

Aicha Mokhtari
USTHB, Algeria
aissani_mokhtari@yahoo.fr

Abstract—To provide dynamic authorizations to users, access control must take into account context. Using this idea, we develop a Contextual Multi-Level Access Control Model based on Description Logic with Default and Exception named $DL - CMLAC_{\delta\epsilon}$. To give a formal representation of this model, we define a non monotonic description logic based system by which we can deal with default and exceptional language called $JClassic_{\delta\epsilon}^+$. It is an extension of $JClassic_{\delta\epsilon}$ in order to introduce disjunction of concepts. $JClassic_{\delta\epsilon}^+$ is expressive enough to be of practical use, it can handle a "weakened kind of disjunction" with the connective *lcs* allowing a tractable subsumption computation. The connective *lcs* has the same properties as the LCS external operation to compute the least common subsumer of two concepts. Connectives of $JClassic_{\delta\epsilon}^+$ are used in a clever way to represent authorization in a default context, an exceptional context and composed context.

Keywords - Description logic; reasoner; disjunction; access control; context.

I. INTRODUCTION

The purpose of access control models is to assign permissions to users. The most interesting would be to have the ability to set dynamic permissions, i.e., context-dependent. Contexts express different types of extra conditions or constraints that control activation of rules expressed in the access control policy. there are several types of context:

- The Temporal context that depends on the time at which the subject is requesting for an access to the system,
- the Spatial context that depends on the subject location,
- the User-declared context that depends on the subject objective (or purpose),
- the Prerequisite context that depends on characteristics that join the subject, the action and the object.
- the Provisional context that depends on previous actions the subject has performed in the system.

We also assume that each organization manages some information system that stores and manages different types of information. To control context activation, each information system must provide the information required to check that conditions associated with the context definition are satisfied or not. The following list gives the kind of information related to the contexts we have just mentioned:

- A global clock to check the temporal context,
- the subject environment and the software and hardware architecture to check the spatial context,
- the subject purpose to check the user-declared context,
- the system database to check the prerequisite context,

- an history of the action carried out, to check the provisional context.

We are interested in modeling the user-declared context, particularly disjunction of several constraints [1].

Literature provides a wide range of access control models and policy languages. One of them is multilevel access control commonly used by military organisations in order to protect the confidentiality of their informations [2]. We propose to develop an access control model inspired from multilevel access control with the introduce of user-declared context to provide dynamic authorization.

In our model, authorization depends not only on classification and clearance levels, but also on the current context, in which user requests a right of access. Each change of context implies a change in permissions.

There are several type of user-declared contexts. It could be the regular one (what we call in our work default context) like it may be an exception to the current context. For example, in current days, each patient in a hospital is treated by his own doctor, but when there is an exception like an emergency, the authorization should change.

Context can be composed of several contexts (constraints). For example, under normal circumstances, the family has the right to visit the patient but when there is a risk of contamination and/or unknown disease, family loses this right. In this paper, we are interested in disjunction of context (*or*).

To provide a formal representation, we use the $JClassic_{\delta\epsilon}^+$ [1] developed by us for this purpose. It is a description logic-based system augmented with two operators δ (for default) and ϵ (for exception) inspired by $AL_{\delta\epsilon}$ description logic [3] and "lcs" (for disjunction).

This kind of non-monotonic reasoning in description logic is not sufficiently developed. Actually, there is no system, in our sense, developed on this kind of reasoning in the web [4].

Description logics are powerful knowledge representation systems providing well-founded and computationally tractable classification reasoning. However, expression of disjunction of concepts has previously been infeasible due to computational cost.

Donini [5] shows that concept disjunction makes subsumption computation co-NP-Complete. However, disjunction is very useful for knowledge representation.

$JClassic_{\delta\epsilon}^+$ is an extension of $JClassic_{\delta\epsilon}$ [6], [7] by the operator of disjunction "lcs". This operator allows us to define context disjunction in access control.

The "lcs" connective has the same properties as the LCS

external (External insofar as LCS is not a connective of the language) operation introduced by Borgida et al. [8], which computes the least common subsumer of two concepts (The Least Common Subsumer of two concept A and B belonging to a language L is the most specific concept in L that subsumes both A and B) [9]. It was introduced by Ventos et al. in Classic to allow disjunction with a reasonable computation [10], [11].

Because of $JClassic_{\delta\epsilon}^+$ has been given an intensional semantics, $JClassic_{\delta\epsilon}^+$ is provided with an intentional semantics (called $\mathcal{CL}_{\delta\epsilon}^+$) based on an algebraic approach. For this, we have first to build an equational system, which highlights the main properties of the connectives. The equational system allows to define axiomatically the notion of LCS operation.

The main operation, the computation of the subsumption relation of $JClassic_{\delta\epsilon}^+$, is used to classify and deduce knowledge. The inheritance relation is used to compute the inherited properties.

In this paper, we first present our description logic system $JClassic_{\delta\epsilon}^+$ and we give definition of "lcs". We illustrate then the use of this reasoner for access control, using a small demonstration to show how authorization can be given to user.

II. RELATED WORK

Several approaches have been proposed to model context in access control. As we have seen earlier, the context makes it possible to express different kinds of constraint.

In RBAC family models, many works were done in this sense, some of them extend RBAC model to deal with access control based on user's location context [12], [13], [14], [15], [16], [17], or temporal context [18]. They suggest to combine concept of role with spatial or temporal condition to obtain contextual roles.

Georgiadis and al. [19] present a team-based access control model that is aware of contextual information associated with activities in application. Hu et al. [20] developed a context-aware access control model for distributed healthcare application. To provide context-aware access control, the model defines the notion of context type and context constraint.

In OrBAC family, context is represented by an argument in the predicate *Permission* [21].

Several extension where done to take into account various types of context such as spatial, temporal and composed context [22], [23].

In these approaches, context is still modeled by an argument in a predicate using some algebra to write context, the authors don't present how the final value of context is calculated. Add to this, first order logic is known to be semi-decidable.

In our approach, context can be atomic or composed context using conjunction or disjunction.

III. $JClassic_{\delta\epsilon}^+$

$JClassic_{\delta\epsilon}^+$ is a non monotonic reasoner based on description logic with default and exception [3], which allows us to deal with default and exceptional knowledge.

The set of connectives of $JClassic_{\delta\epsilon}^+$ is the union of the set of connectives of $\mathcal{AL}_{\delta\epsilon}$ [3] presented in [7], [24], [25] and the connective "lcs".

The connective δ intuitively represents the common notion of default. For instance, having Animal as a conjunction with the concept δFly in the definition of the concept **Bird** states that birds generally fly.

The connective ϵ is used to represent a property that is not present in the description of the concept or of the instance but that should be. For instance, the definition of **Penguin** in $JClassic_{\delta\epsilon}^+$ is $Penguin \sqsubseteq Bird \sqcap Fly^\epsilon$. The Fly^ϵ property expresses the fact that fly should be in the definition of Penguin since it is a bird. The presence of Fly^ϵ in the definition of Penguin makes it possible to classify Penguin under the concept Bird.

Formally, the subsumption relation uses an algebraic semantics. The main interest of this approach is the introduction of the definitional point of view of default knowledge: from the definitional point of view, default knowledge can be part of concept definition whereas from the inheritance is only considered as a weak implication. A map between the definition of concept and its inherited properties is done with the calculation of its normal form. This combining of definitional and inheritance levels improves the classification process.

In this section, we first present the syntax of our system, we then give details about its algebraic semantic.

A. Syntax of $JClassic_{\delta\epsilon}^+$

The set of connectives of $JClassic_{\delta\epsilon}^+$ is the union of the set of connectives of $\mathcal{CL}_{\delta\epsilon}$ [6] and the connective *lcs*. $JClassic_{\delta\epsilon}^+$ is defined using a set \mathbf{R} of primitive roles, a set \mathbf{P} of primitive concepts, the constant \perp (Bottom) and \top (Top) and the following syntax rule (C and D are concepts, P is a primitive concept, R is a primitive role).

δ and ϵ are unary connectives, \sqcap is a binary conjunction connective and \forall enables universal quantification on role values. The Terminological language is given in Table 1.

B. Semantic of $JClassic_{\delta\epsilon}^+$

We endow $JClassic_{\delta\epsilon}^+$ with an intentional algebraic semantic denoted $\mathcal{CL}_{\delta\epsilon}^+$.

This framework covers the different aspects of the formal definition of concepts and subsumption in our language. The calculating of denotations of concepts in $\mathcal{CL}_{\delta\epsilon}^+$ is used in computing subsumption in the algorithm $Sub_{\delta\epsilon}^+$. $\mathcal{CL}_{\delta\epsilon}^+$ allows first to show that $Sub_{\delta\epsilon}^+$ is correct and complete and secondly to give a formal characterization of calculation of subsumption used in the implementation of $JClassic_{\delta\epsilon}^+$.

Subsumption is considered from two points of view:

- A descriptive point of view: it consists on the comparison of terms through an equational system;
- A structural point of view: it consists on a comparison of normal forms of concept

1) *EQ*: an equational system for $JClassic_{\delta\epsilon}^+$: In order to serve as the basis for the definition of an algebraic semantics, an equational system EQ is defined. From a descriptive point of view, the calculation of subsumption consists on the comparison of terms through the equational system EQ. This

$C, D \rightarrow \top$	the most general concept
\perp	the most specific concept
P	primitive concept
$C \sqcap D$	concept conjunction
$\neg P$	negation of primitive concept (This restriction to primitive concept in the negation is a choice to avoid the untractability)
$\forall r : C$	C is a value restriction on all roles R
R AT-LEAST n	cardinality for R (minimum)
R AT-MOST n	cardinality for R (maximum)
δC	default concept
C^ϵ	exception to the concept
$C \text{ lcs } D$	concept disjunction

TABLE I
SYNTAX OF $JClassic_{\delta\epsilon}^+$

system fixes the main properties of the connectives and is used to define an equivalence relation between terms and then to formalize the subsumption relationship.

$\forall A, B, C \in JClassic_{\delta\epsilon}^+$:

01: $(A \sqcap B) \sqcap C = A \sqcap (B \sqcap C)$

02: $A \sqcap B = B \sqcap A$

03: $A \sqcap A = A$

04: $\top \sqcap A = A$

05: $\perp \sqcap A = \perp$

06: $(\forall R : A) \sqcap (\forall R : B) = \forall R : (A \sqcap B)$

07: $\forall R : \top = \top$

08: R AT-LEAST m \sqcap R AT-LEAST n = R AT-LEAST

maxi(m,n)

09: R AT-LEAST 0 = \top

10: R AT-MOST m \sqcap R AT-MOST n = R AT-MOST

mini(m,n)

11: R AT-MOST 0 = $\forall R : \perp$

12: R AT-LEAST m \sqcap R AT-MOST n (if $n \leq m$).

13: $(A \text{ lcs } B) \text{ lcs } C = A \text{ lcs } (B \text{ lcs } C)$

14: $A \text{ lcs } B = B \text{ lcs } A$

15: $A \text{ lcs } A = A$

16: $A \text{ lcs } \top = \top$

17: $A \text{ lcs } \perp = A$

18: $(\delta A)^\epsilon = A^\epsilon$

19: $\delta(A \sqcap B) = (\delta A) \sqcap (\delta B)$

20: $A \sqcap \delta A = A$

21: $A^\epsilon \sqcap \delta A = A^\epsilon$

22: $\delta \delta A = \delta A$

23: $(A^\epsilon)^\epsilon = \delta A$

Axioms 01 to 12 are classical; they concern description logic connectives properties [26], [27]. Axioms 13 to 17 concern the connective "lcs". The following ones correspond to $\mathcal{AL}_{\delta\epsilon}$ connectives properties[3], i.e., properties of δ and ϵ connectives.

Descriptive Subsumption:

We denote \sqsubseteq_d for descriptive subsumption. \sqsubseteq_d is a partial order relation on terms. Equality (modulo the axioms of EQ) between two terms is denoted $=_{EQ}$. $=_{EQ}$ is a congruence relation which partitions the set of terms, i.e., $=_{EQ}$ allows to form equivalence classes between terms. We define the

descriptive subsumption using the congruence relation and conjunction of concepts as follow:

Definition 1: (Descriptive Subsumption)

Let C and D two terms of $JClassic_{\delta\epsilon}^+$, $C \sqsubseteq_d D$, i.e., D subsume descriptively C, iff $C \sqcap D =_{EQ} C$.

From an algorithmic point of view, terms are not easily manipulated through subsumption. We adopt a structural point of view closer to the algorithmic aspect of computing subsumption. This allows us to first formalize calculation of subsumption in the implementation of $JClassic_{\delta\epsilon}^+$ and secondly to endow $JClassic_{\delta\epsilon}^+$ with an intensional semantics.

To define the subsumption relation between two concepts using their description, we need to compare them. For this, concepts are characterized by a normal form of their properties rather than by the set of their instances.

2) *Normal Form of concept:* We present in this section the structural point of view for the subsumption in $JClassic_{\delta\epsilon}^+$. This point of view has two main advantages: it is very close to the algorithmic aspects and is a formal framework to validate the algorithmic approach, which is not the case description graph.

We define a structural concept algebra $\mathcal{CL}_{\delta\epsilon}^+$, which is used to give an intensional semantic in which concepts are denoted by the normal form of their set of properties. The structural point of view of subsumption consist then to compare the normal forms derived by applying a homomorphism from set of terms of $JClassic_{\delta\epsilon}^+$ to elements of $\mathcal{CL}_{\delta\epsilon}^+$.

$\mathcal{CL}_{\delta\epsilon}^+$: an intensional semantic for $JClassic_{\delta\epsilon}^+$

From the class of CL-algebra, we present a structural algebra $\mathcal{CL}_{\delta\epsilon}^+$, which allows to endow $JClassic_{\delta\epsilon}^+$ with an intensional semantic.

Element of $\mathcal{CL}_{\delta\epsilon}^+$ are the canonical intentional representation of terms of $JClassic_{\delta\epsilon}^+$ (i.e., Normal form of the set of their properties). We call an element of $\mathcal{CL}_{\delta\epsilon}^+$ normal forms.

Definition of $\mathcal{CL}_{\delta\epsilon}^+$ means definition of a homomorphism h, which allows to associate an element of $\mathcal{CL}_{\delta\epsilon}^+$ to a term of $JClassic_{\delta\epsilon}^+$.

Using the equational system, we calculate for each concept

a structural denotation, which is a single normal form of this concept. The calculation of a normal form from a description of a concept can be seen as a result of term “rewriting” based on the equational system EQ.

The normal form of a concept defined with description T (noted $\text{nf}(T)$) is a pair $\langle t_\theta, t_\delta \rangle$ where t_θ contains strict properties of T and t_δ the default properties of T.

t_θ and t_δ are 6-uplet of the form $(\text{dom}, \text{min}, \text{max}, \pi, r, \epsilon)$ with:

dom: is a set of Individuals, if the description contain the property ONE-OF else the symbol UNIV.

min (resp. max): is a real, if the description contain the property Min (resp. Max) else the symbol MIN-R (resp. MAX-R).

π : is a set of primitive concept in description T.

r: have the form $\langle R, \text{fillers}, \text{least}, \text{most}, c \rangle$ where:

R: the name of Role.

fillers: set of Individuals, if the description contain the property R Fills, else \emptyset .

least (resp. most): is an integer, if the description contain AT-LEAST (resp. AT-MOST), else 0 (resp. NOLIMIT).

c: is the normal form of C, if the description contain the property $\forall R : C$.

ϵ : set of 6-uplet with the form $(\text{dom}, \text{min}, \text{max}, \pi, r, \epsilon)$.

Example: The normal form of concept $C \equiv A \sqcap \delta B$ is:

$\text{fn}(C) = (\langle \text{Univ}, \text{Min} - R, \text{Max} - R, \{A\}, \emptyset, \emptyset \rangle, \langle \text{Univ}, \text{Min} - R, \text{Max} - R, \{A, B\}, \emptyset, \emptyset \rangle)$.

The interpretation of connectors and constants of $\mathcal{CL}_{\delta\epsilon}^+$ is given in Table 2. b_0 is a constant used as a denotation of \perp [6].

Structural Subsumption:

Two terms C and D of $J\text{Classic}_{\delta\epsilon}^+$ are structurally equivalent iff their normal forms are equal. We denote \sqsubseteq_s for structural subsumption. \sqsubseteq_s is a partial order relation.

The structural equality of two terms of $J\text{Classic}_{\delta\epsilon}^+$ is noted $=_{CL}$. $=_{CL}$ is a congruence relation as $=_{EQ}$ in descriptive subsumption.

We define the structural subsumption using the congruence relation and conjunction of concepts as follow:

Definition 2: (Structural Subsumption)

Let C and D two terms of $J\text{Classic}_{\delta\epsilon}^+$, $C \sqsubseteq_s D$; i.e., D subsume structurally C, iff $C \sqcap D =_{CL} C$.

Theorem 1: (Equivalency between descriptive subsumption and structural subsumption)

Let C and D two terms of $J\text{Classic}_{\delta\epsilon}^+$, $C \sqsubseteq_s D \Leftrightarrow C \sqsubseteq_d D$.

To infer new knowledge in this system, the subsumption relation is the main operation. In the next section, we outline the subsumption algorithm handling defaults and exceptions named $\text{Sub}_{\delta\epsilon}$.

IV. INFERENCE IN $J\text{Classic}_{\delta\epsilon}^+$

There are several reasoning services to be provided by a DL- system. We concentrate our work on the following

basic ones, which are classification of concepts (TBox) and instance checking (ABox). These two services basically use the subsumption relation.

A. The Subsumption Relation

Borgida [8] defines the subsumption based on a set theoretic interpretation as follow: “The concept C subsume D, if and only if the set of instances of C include or is equal to a set of instances of D”.

However, the general principle of computing subsumption between two concepts is to compare their sets of properties, not their sets of instances.

For this, we use an intensional semantics which is closer to the algorithmic aspects of computing subsumption, and this by defining a normal form of description called descriptive normal form.

Algorithm of Computing Subsumption $\text{Sub}_{\delta\epsilon}^+$

$\text{Sub}_{\delta\epsilon}^+$ is an algorithm of computing subsumption of the form Normalization- Comparison. It consists of two steps, first, the normalization of description, and then a syntactic comparison of the obtained normal forms.

Let C and D be two terms of $J\text{Classic}_{\delta\epsilon}^+$. To answer the question “Is C subsumed by D?” we apply the following procedure. The normal forms of C and “ $C \sqcap D$ ” are calculated with the procedure of normalisation.

There are two steps in the comparison. We compare the strict parts of the two concepts. If these are equal, then we compare the default parts. If the two normal forms are equal, the algorithm returns “Yes”. It returns “No” otherwise.

Algorithm 1 Algorithm $\text{Sub}_{\delta\epsilon}^+$

Require: C and D two description of concepts of $J\text{Classic}_{\delta\epsilon}^+$
Ensure: Response “Yes” or “No” to question “Is C subsumed by D?”

```

{Compute normal forms}
fn(C) ← Normalization(C)
fn(C  $\sqcap$  D) ← Normalization(C  $\sqcap$  D)
{Treatment of bottom}
if fn(C)= $b_0$  then
    Response ← “Yes”
else
    if fn(C  $\sqcap$  D)= $b_0$  then
        Response ← “No”
    else
        {Comparison of the obtained normal forms}
        Compar(fn(C) $_\theta$ , fn(C $\sqcap$  D) $_\theta$ , rep1)
        if rep1=“Yes” then
            Compar(fn(C) $_\delta$ , fn(C $\sqcap$  D) $_\delta$ , rep1)
            Response ← rep2
        else
            Response ← “No”
        end if
    end if
end if

```

$JClassic_{\delta\epsilon}$	$\mathcal{CL}_{\delta\epsilon}^+$
\top	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset) \rangle$
P	$\langle (UNIV, MIN-R, MAX-R, P, \emptyset, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset) \rangle$
ONE-OF E	$\langle (E, MIN-R, MAX-R, P, \emptyset, \emptyset), (E, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset) \rangle$
MIN u	$\langle (UNIV, u, MAX-R, \emptyset, \emptyset, \emptyset), (UNIV, u, MAX-R, \emptyset, \emptyset, \emptyset) \rangle$
MAX u	$\langle (UNIV, MIN-R, u, \emptyset, \emptyset, \emptyset), (UNIV, MIN-R, u, \emptyset, \emptyset, \emptyset) \rangle$
$\forall R : C(C \neq \top \text{ et } C \neq \perp)$	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, \emptyset, 0, c_{\theta.dom} , c \rangle\}, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, \emptyset, 0, c_{\theta.dom} , c \rangle\}, \emptyset) \rangle$
$\forall R : C \text{ et } C \equiv \perp$	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, \emptyset, 0, 0, b_0 \rangle\}, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, \emptyset, 0, 0, b_0 \rangle\}, \emptyset) \rangle$
$\forall R : C \text{ et } C \equiv \top$	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset) \rangle$
$R FILLSE(E \neq \emptyset)$	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, E, E , NOLIMIT, t \rangle\}, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, E, E , NOLIMIT, t \rangle\}, \emptyset) \rangle$
R FILLS \emptyset	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset) \rangle$
$RAT - LEAST n(n \neq 0)$	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, \emptyset, n, NOLIMIT, t \rangle\}, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, \emptyset, n, NOLIMIT, t \rangle\}, \emptyset) \rangle$
R AT-LEAST 0	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset) \rangle$
$RAT - MOST n(n > 0)$	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, \emptyset, 0, n, t \rangle\}, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, \emptyset, 0, n, t \rangle\}, \emptyset) \rangle$
R AT-MOST 0	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, \emptyset, 0, 0, b_0 \rangle\}, \emptyset), (UNIV, MIN-R, MAX-R, \emptyset, \{\langle R, \emptyset, 0, 0, b_0 \rangle\}, \emptyset) \rangle$
$C \sqcap D$	$c \otimes d$
$C \text{ lcs } D$	$c \text{ LCS } d$
δC	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, \emptyset), c_{\delta} \rangle$
C^{ϵ}	$\langle (UNIV, MIN-R, MAX-R, \emptyset, \emptyset, c_{\delta}), (c_{\delta.dom}, c_{\delta.max}, c_{\delta.min}, c_{\delta\pi}, c_{\delta r}, c_{\delta\epsilon} \cup c_{\delta}) \rangle$
\perp	b_0

TABLE II
INTERPRETATION OF CONNECTORS AND CONSTANTS OF $\mathcal{CL}_{\delta\epsilon}^+$

The completeness, correctness and the polynomial computation of $JClassic_{\delta\epsilon}$ have been proved in [7].

B. Classification of concept

The classification of concepts is an operation which inserts a concept to the most appropriate place in the hierarchy. The classification process allows to find subsumption relations between concepts in the taxonomy (hierarchy) and insert new concept in the hierarchy.

In the $JClassic_{\delta\epsilon}^+$ reasoner, the classification of a concept consists of two phases: first phase is to find the most specific concepts that subsume the concept C (the concept C is to classify), they are called subsumers (SPS). The second phase search the most general concepts than C, we call them subsumed (GSP), it also establish new relations between the concept to classify C, its SPS and its GSP.

The classification process is triggered when we create a new concept (primitive or defined).

C. Instance recognition

The recognition of instances is to find for a given individual the most specific concepts which it may be an instance.

We used the method to achieve Abstraction-Classification mechanism instantiation of concepts.

The method Abstraction - Classification

This method allows the instantiation of the individual, it consists of two phases:

Abstraction: calculates the abstract concept of the individual containing all the information in the form of an abstract defined concept.

Classification: is to find the abstract concept of SPS, the SPS corresponds to the direct instances of individual, in other words: To determine whether an object O is instance of a concept C, we calculate the abstract concept AO, we then check if C subsumes AO. If so, we deduce that O is an instance of concept C, else O is not an instance of concept C.

Ex: if an individual named "Sara" eats only plants, the reasoner determine that Sara is an instance of concept VEG-ETARIAN.

D. Inheritance relation

The inheritance relation allows to compute the inherited properties of a concept. These properties are the basic ones in inferential systems.

The inheritance relation serves as a basis for retrieving the inherited properties, it also helps in distinguishing strict and

default inherited properties and answering questions concerning conflicts and consistency.

The main task of the inheritance relation is to retract exceptions from the denotation of a concept (the two ϵ parts of denotation). The inheritance-form scenario of a concept C is:

1- Replace each exception at an even level by a default in the denotation of C .

2- For each role of C , recursively call inheritance with the role value restriction.

3- Suppress P (resp. P°) in $c_{\delta\pi}$ if P° (resp. P) is in $c_{\phi\pi}$. The resulting denotation is called the inheritance form of C (The set of primitive concepts P is complemented with a new set P° in order to denote negation. There is no axiom that relates a primitive concept to its negation. This set P° is then theoretically necessary, but transparent for the user).

Using this relation, we deduce the inherited properties, and type them between strict and default ones. The inherited properties are those found in the inheritance form of the concept. The strict ones (resp. the default ones) are those in the strict (resp. default) part.

Algorithm 2 Inheritance Map

```

inheritance:  $\mathcal{CL}_{\delta\epsilon}^+ \rightarrow \mathcal{CL}_{\delta\epsilon}^+$ , such that
inheritance(a)=
res  $\leftarrow \prec (a_{\theta\pi}, \emptyset, \emptyset), (a_{\delta\pi}, \emptyset, \emptyset) \succ$ 
for all  $y \in a_{\theta\epsilon} \cup a_{\delta\epsilon}$  do
  res  $\leftarrow$  res  $\cup$  transform( $y, a_{\theta\epsilon}$ )
end for
for all  $\prec r, p \succ \in a_{\theta r}$  do
  res  $\leftarrow$  res  $\cup \prec (\emptyset, \prec r, \text{inheritance}(p) \succ, \emptyset), (\emptyset, \emptyset, \emptyset) \succ$ 
end for
for all  $\prec r, p \succ \in a_{\delta r}$  do
  res  $\leftarrow$  res  $\cup \prec (\emptyset, \emptyset, \emptyset), (\emptyset, \prec r, \text{inheritance}(p) \succ, \emptyset) \succ$ 
end for
let res be  $\prec (res_{\theta\pi}, res_{\theta r}, res_{\theta\epsilon}), (res_{\delta\pi}, res_{\delta r}, res_{\delta\epsilon}) \succ$ 
for all  $x \in res_{\theta\pi}$  do
  suppress  $x^\circ$  from  $res_{\delta\pi}$ 
end for
for all  $x^\circ \in res_{\theta\pi}$  do
  suppress  $x$  from  $res_{\delta\pi}$ 
end for
return res

```

We detailed in the next section the connective "lcs" and the operation LCS , by which we compute normal form of A lcs B (A and B are concepts).

V. THE COMPUTATION OF "LCS"

The least common subsumer has been introduced in description logic by Borgida et al. [8] as an external operation to compute the LCS of two concepts.

The LCS of two concepts A and B belonging to a language L is the most specific concept in L that subsumes both A and B .

Definition 3: Let L a terminological language, \sqsubseteq the notation of subsumption relation in L

$LCS: L \times L \rightarrow L$

$LCS(A,B) \rightarrow C \in L$ iff:

$A \sqsubseteq C$ and $B \sqsubseteq C$ (C subsume both A and B),

$\nexists D \in L$ such that $A \sqsubseteq D, B \sqsubseteq D$ and $D \sqsubseteq C$ (i.e., there is no common subsumers to A and B , which is subsumed strictly by C)

The next algorithm is to compute the LCS where input are the normal form of two concepts A_1 and A_2 and the output is the LCS of A_1 and A_2 .

Let a and b two normal forms A and B with a and $b \neq b_0$ (b_0 is the normal form of \perp).

Algorithm 3 LCS

Require: $a = \prec a_\theta, a_\theta \succ$ and $b = \prec b_\theta, b_\theta \succ$ two normal forms of A and B .

Ensure: $c = \prec c_\theta, c_\theta \succ$ the normal form of $LCS(A,B)$

```

 $c_{\theta\pi} \leftarrow a_{\theta\pi} \cap b_{\theta\pi}$ 
 $c_{\theta r} \leftarrow \emptyset$ 
for all  $\prec r, d \succ \in a_{\theta r}$  do
  if  $\exists \prec r, e \succ \in b_{\theta r}$  then
     $f \leftarrow LCS(d,e)$ 
     $c_{\theta r} \leftarrow c_{\theta r} \cup \prec r, f \succ$ 
  end if
end for
 $c_{\theta\epsilon} \leftarrow a_{\theta\epsilon} \cap b_{\theta\epsilon}$ 
 $c_{\delta\pi} \leftarrow a_{\delta\pi} \cap b_{\delta\pi}$ 
 $c_{\delta r} \leftarrow \emptyset$ 
for all  $\prec r, d \succ \in a_{\delta r}$  do
  if  $\exists \prec r, e \succ \in b_{\delta r}$  then
     $f \leftarrow LCS(d,e)$ 
     $c_{\delta r} \leftarrow c_{\delta r} \cup \prec r, f \succ$ 
  end if
end for
 $c_{\delta\epsilon} \leftarrow a_{\delta\epsilon} \cap b_{\delta\epsilon}$ 

```

$JClassic_{\delta\epsilon}^+$ can be used in differents application. We will use it to formalize our contextual multilevel access control model, in which the context could be in different forms.

VI. APPLICATION TO ACCESS CONTROL

To show how we can use our description logic-based system and how we can infer new knowledge, we define a knowledge base adapted to formalize a dynamic access control model named $DL - CMLAC_{\delta\epsilon}$ (Contextual Multi-Level Access Control Model based on Description Logic with Default and Exception).

In this model, authorization to subject is assigned depending on context. We consider first that the context is by default normal, and we represent it using the operator of default (δ). Then, each change of context is considered as an exception to the current context, this change is represented by the operator of exception (ϵ). We give, as an example, one ABox to show how authorization can be deduced.

We are interested in one kind of policy, which is a multilevel access control commonly used by military organizations in order to protect the confidentiality of their informations [2].

In multilevel access control model, a subject s can access to an object o only if its clearance level is greater than or equal to the classification level of the object.

To allow the policy designer to define a security policy independently of the implementation, we introduce an abstract level.

Subject and Object are respectively abstracted into Role and View. A role is a set of subjects to which the same security rule apply and similarly, a view is a set of objects to which the same security rule apply. For example, the subject "John" plays the role of "Doctor" in the organization "Service of Pediatrics" and the view "Medical record" corresponds to the object "Medical record of patient". A clearance level is assigned to the role and a classification level is assigned to a View. An abstract authorization is assigned to a role on a view in a given context if its clearance level is greater than or equal to the classification level of the view.

The concrete authorization is derived from the abstract one depending on context.

In our approach, we take into account the context by considering the following postulate.

Postulate 1 (Normal context). *By default, the context is normal (usual context).*

Postulate 2. *All actions that are not permitted are prohibited.*

Figure 1 describes the general architecture of our access control model.

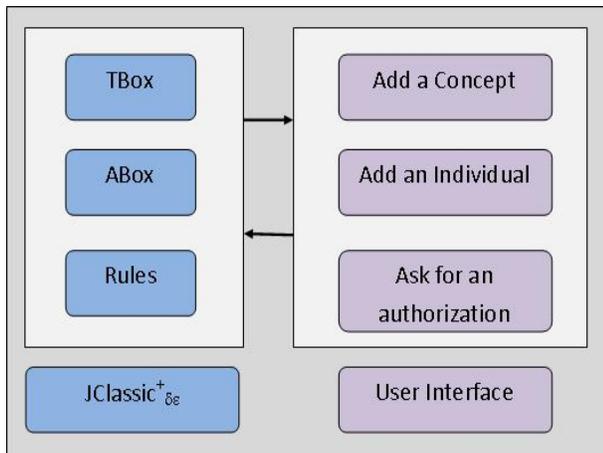


Fig. 1. Architecture of $DL - CMLAC_{\delta\epsilon}$

A. TBox

We now conceptualize access control model by a DL knowledge base capturing its characteristics, including the context with the use of defaults (δ) and exceptions (ϵ), using the Bell

and La Padula model [28]. We define then TBox and ABox axioms with examples to illustrate their content and use.

We define a DL knowledge base \mathcal{K} . The alphabet of \mathcal{K} includes the following atomic concepts: Subject, Object, Role, View, LevelR and LevelV. The TBox includes the following axioms.

- **Role attribution axiom:**

$$Subject \sqsubseteq \top$$

$$Role \sqsubseteq \top$$

$$Employ \sqsubseteq EmployS.Subject \sqcap EmployR.Role$$

It defines the relationship between subject and role, where:

EmployS and EmployR are binary relations such as:

- EmployS : EmployS links the concept Employ to the concept Subject

- EmployR : EmployR links the concept Employ to the concept Role.

- **View definition axiom:**

$$Object \sqsubseteq \top$$

$$View \sqsubseteq \top$$

$$Use \sqsubseteq UseO.Object \sqcap UseV.View$$

It defines relationship between object and view, where:

UseO and UseV are binary relations such as:

- UseO : UseO links the concept Use to the concept Object.

- UseV : UseV links the concept Use to the concept View.

- **Classification definition axiom:**

$$LevelV \sqsubseteq \top$$

$$Attribute \sqsubseteq AttributeV.View \sqcap AttributeL.LevelV$$

It defines relationship between the view and its classification level, where:

AttributeV and AttributeL are binary relations such as:

- AttributeV : AttributeV links the concept Attribute to the concept View.

- AttributeL : AttributeL links the concept Attribute to the concept LevelV.

- **Clearance definition axiom:**

$$LevelL \sqsubseteq \top$$

$$Assign \sqsubseteq AssignR.Role \sqcap AssignL.LevelR$$

It defines relationship between the view and its classification level, where:

AssignR and AssignL are binary relations such as:

- AssignR : AssignR links the concept Assign to the concept Role.

- AssignL : AssignL links the concept Assign to the concept LevelR.

Depending on the model of Bell and La Padula [28], there are two types of authorization, one for reading permission and another one for writing permission. And because we have two levels in our model: abstract and concrete level, we will give axioms for abstract permission (reading and writing) and axioms for concrete permission (reading and writing).

To define a default permission, we use the following axioms.

- **Reading Permission attribution axiom:** defines the relation between role and view. A default reading permission is given to role **R** on a view **V** when its clearance level is greater than or equal to the classification level of the view.

$$\delta RPermission \sqsubseteq RPermission.R.Role \sqcap RPermission.V.View \sqcap Attribute \sqcap Assign \sqcap LevelR$$

Where:

RPermissionR and RPermissionV are binary relations such as:

- RPermissionR : RPermissionR links the concept RPermission to the concept Role.
- RPermissionV : RPermissionV links the concept RPermission to the concept View.
- LevelR \sqsubseteq haslevel At-least LevelV

- **Writing Permission attribution axiom:** defines the relation between role and view. A default writing permission is given to role **R** on a view **V** when its clearance level is less than or equal to the classification level of the view.

$$\delta WPermission \sqsubseteq WPermission.R.Role \sqcap WPermission.V.View \sqcap Attribute \sqcap Assign \sqcap LevelR$$

Where:

WPermissionR and WPermissionV are binary relations such as:

- WPermissionR : WPermissionR links the concept WPermission to the concept Role.
- WPermissionV : WPermissionV links the concept WPermission to the concept View.
- LevelR \sqsubseteq haslevel At-most LevelV

A concrete permission is expressed with the next axioms.

- **Concrete Reading Permission axiom:**

$$Is - Rpermitted \sqsubseteq Is - Rpermitted.S.Subject \sqcap Is - Rpermitted.O.Object$$

A concrete reading permission is given to subject **S** on an object **O**, where:

Is-RpermittedS, Is-RpermittedO are binary relations such as:

- Is-RpermittedS : Is-RpermittedS links the concept Is-Rpermitted to the concept Subject.
- Is-RpermittedO : Is-RpermittedO links the concept Is-Rpermitted to the concept Object.

- **Concrete Writing Permission axiom:**

$$Is - Wpermitted \sqsubseteq Is - Wpermitted.S.Subject \sqcap Is - Wpermitted.O.Object$$

A concrete writing permission is given to subject **S** on an object **O**, where:

Is-WpermittedS, Is-WpermittedO are binary relations such as:

- Is-WpermittedS : Is-WpermittedS links the concept Is-Wpermitted to the concept Subject.
- Is-WpermittedO : Is-WpermittedO links the concept Is-Wpermitted to the concept Object.

Definition of rules of security:

$$\delta Is - Rpermitted \sqsubseteq Employ \sqcap Use \sqcap \delta RPermission$$

$$\delta Is - Wpermitted \sqsubseteq Employ \sqcap Use \sqcap \delta WPermission$$

- If a subject **S** is employed in a role **R** (*Employ*), and if there is a relation between an object **O** and a view **V** (*Use*), and if we have a default reading permission (resp. default writing permission) relation between role **R** and a view **V** ($\delta RPermission$) (resp. ($\delta WPermission$)), we deduce that a subject **S** is by default permitted to perform action of reading (resp. writing) on object **O** ($\delta Is - Rpermitted$) (resp. $\delta Is - Wpermitted$), and because $Is - Rpermitted \sqsubseteq \delta Is - Rpermitted$ (resp. $Is - Wpermitted \sqsubseteq \delta Is - Wpermitted$) (a concrete permission can be deduced from a default permission), we can finally say that a subject **S** is permitted to perform action of reading (resp. writing) on object **O**.

$$Is - Rpermitted^\epsilon \sqsubseteq Employ \sqcap Use \sqcap RPermission^\epsilon$$

$$Is - Wpermitted^\epsilon \sqsubseteq Employ \sqcap Use \sqcap WPermission^\epsilon$$

- By cons, if we have an exception on a reading permission concept wrote $RPermission^\epsilon$ (resp. writing permission concept wrote $WPermission^\epsilon$), we say that we have an exception on a concept $Is - Rpermitted$ wrote $Is - Rpermitted^\epsilon$ (resp. exception on a concept $Is - Wpermitted$ wrote $Is - Wpermitted^\epsilon$), and because $Is - Rpermitted \not\sqsubseteq Is - Rpermitted^\epsilon$ (resp. $Is - Wpermitted \not\sqsubseteq Is - Wpermitted^\epsilon$) (a concrete permission can not be deduced from an exceptional permission), we can deduce that a subject **S** is prohibited to perform action of reading (resp. writing) on object **O**.

B. The ABox

It contains statement about individuals. We could have many ABox for one TBox depending on applications. We illustrate this in the next section, we show how a security policy can be handled by our tool and how we can infer authorizations.

- Using instances of Table 3, the system cannot infer that **Jean** has the **default permission to read** the **PS1** because the classification level of the role Secretary which is played by Jean is less than the clearance level of the view Project-statistics. And because the default permission cannot be deduced, the concrete permission cannot also be deduced.

Suppose now that the assistant is absent, and the director needs statistics of the project.

The context now is different, and it is considered as an exception to the default one. We can give the secretary a temporary permission by changing his classification level, this change is only valid in this context.

We can now deduce that **Jean** has the **default permission to read** the **PS1** because the context *Absence of the Assistant* is true.

Then we add this instance to the ABox : $\delta Permission(P1)$.

Where:

$$\delta RPermission(P1) \sqsubseteq RPermission.V.View(Project - statistics) \sqcap RPermission.R.Role(Secretary) \sqcap Attribute(At2) \sqcap Assign(As4) \sqcap LevelR$$

ABox
<i>Role(Director);</i>
<i>Role(Assistant);</i>
<i>Role(Secretary);</i>
<i>Subject(Adam);</i>
<i>Subject(Sara);</i>
<i>Subject(Jean);</i>
<i>View(Project-contract);</i>
<i>View(Project-statistics);</i>
<i>View(Project-description);</i>
<i>Object(PC1);</i>
<i>Object(PS1);</i>
<i>Object(PD1);</i>
<i>LevelR(Secret);</i>
<i>LevelR(Confidential);</i>
<i>LevelR(Public);</i>
<i>LevelV(Secret);</i>
<i>LevelV(Confidential);</i>
<i>LevelV(Public);</i>
<i>Employ(E1) ⊆ EmployS.Subject(Adam) ⊓ EmployR.Role(Director);</i>
<i>Employ(E2) ⊆ EmployS.Subject(Sara) ⊓ EmployR.Role(Assistant);</i>
<i>Employ(E3) ⊆ EmployS.Subject(Jean) ⊓ EmployR.Role(Secretary);</i>
<i>Use(U1) ⊆ UseO.Object(PC1) ⊓ UseV.view(Project – contract);</i>
<i>Use(U2) ⊆ UseO.Object(PS1) ⊓ UseV.view(Project – statistical);</i>
<i>Use(U3) ⊆ UseO.Object(PD1) ⊓ UseV.view(Project – description);</i>
<i>Attribute(At1) ⊆ AttributeV.View(Project – contract) ⊓ AttributeL.LevelV(Secret);</i>
<i>Attribute(At2) ⊆ AttributeV.View(Project – statistics) ⊓ AttributeL.LevelV(Confidential);</i>
<i>Attribute(At3) ⊆ AttributeV.View(Project – description) ⊓ AttributeL.LevelV(Public);</i>
<i>Assign(As1) ⊆ AssignR.Role(Director) ⊓ AssignL.LevelR(Secret);</i>
<i>Assign(As2) ⊆ AssignR.Role(Assistant) ⊓ AssignL.LevelR(Confidential);</i>
<i>Assign(As3) ⊆ AssignR.Role(Secretary) ⊓ AssignL.LevelR(Public);</i>

TABLE III
ABox

and, $Assign(As4) \sqsubseteq AssignR.Role(Secretary) \sqcap AssignL.LevelR(Confidential)$

- **Access control if the context Absence of the Assistant is true:** Suppose that user Jean want to read the PS1, can he obtain that privilege?

We know that:

- Jean plays the role of Secretary: $Employ(E3)$;
- and, PS1 is an object used in the view Project-statistics: $Use(U2)$;
- and, in this context, Secretary has a clearance level equal to Confidential: $Assign(As4)$;
- and, Project-statistics has a classification level equal to Confidential: $Attribute(At2)$;
- and finally, by default, each person who plays the role of Secretary is permitted to consult Project-statistics when the assistant is absent: $\delta RPermission(P1)$.

Formally, we write:

$$Employ(E1) \sqcap Use(U1) \sqcap \delta RPermission(P1)$$

Using security rules, we can deduce that the preceding proposition subsumes $\delta Is - Rpermitted(I1)$.

Where:

$$\begin{array}{l} Is - Rpermitted(I1) \quad \sqsubseteq \quad Is - \\ RpermittedS.Subject(Sara) \quad \sqcap \quad Is - \\ RpermittedO.Object(DB - Exam) \end{array}$$

And because $Is - Rpermitted(I1) \sqsubseteq \delta Is - Rpermitted(I1)$, we can deduce that Jean is permitted

to read PS1 if the assistant is absent.

- **suppose that the assistant is absent and a substitute was brought:** can Jean read PS1?

In the context **Assistant absent + substitute present**, the system deduce a new instance P2 and we add to the ABox the next rule:

$$Permission(P1)^\epsilon \sqsubseteq \delta Permission(P2)$$

We know that:

- Jean plays the role of Secretary: $Employ(E3)$;
- and, PS1 is an object used in the view Project-statistics: $Use(U2)$;
- and, in this context, Secretary has a clearance level equal to Confidential: $Assign(As4)$;
- and, Project-statistics has a classification level equal to Confidential: $Attribute(At2)$;
- and finally, by default, each person who plays the role of Secretary is permitted to consult Project-statistics when the assistant is absent and there is a new substitute: $\delta RPermission(P2)$.

We obtain:

$$\begin{array}{l} Employ(E3) \sqcap Use(U2) \sqcap \delta Permission(P2) \\ \equiv Employ(E3) \sqcap Use(U2) \sqcap \delta Permission(P1)^\epsilon \\ \text{We know that } A^\epsilon \equiv \delta A^\epsilon, \text{ we obtain:} \\ \equiv Employ(E3) \sqcap Use(U2) \sqcap Permission(P1)^\epsilon \end{array}$$

Using security rules, we can deduce that the precedent

proposition subsumes $Is - permitted(I1)^\epsilon$.

And, because $Is - permitted(I1) \not\sqsubseteq Is - permitted(I1)^\epsilon$, we cannot deduce $Is-permitted(I1)$. Therefore Jean is not permitted to read PS1 when there is a substitute to the absent assistant.

Our policy language allows us to have more than one exception in a context. Exception at an even level cancel the effects of exceptions and therefore infers the property by default [3].

Suppose that we have a disjunction of context, for example "absence of assistant or absence of assistant with substitute present", here we can use the connective "lcs" to deduce permission

- **lcs(absence of assistant, absence of assistant with substitute present)**: Suppose that user Jean wants to read PS1; can he obtain that privilege?

We know that:

- Jean plays the role of Secretary: $Employ(E3)$;
- and, PS1 is an object used in the view Project-statistics: $Use(U2)$;
- and, in this context, Secretary has a clearance level equal to Confidential: $Assign(As4)$;
- and, Project-statistics has a classification level equal to Confidential: $Attribute(At2)$;
- and we have the two previous permissions $Permission(P1)$ and $Permission(P2)$, defined respectively for the context (absence of assistant) and context (absence of assistant with substitute present).

We obtain:

$$\begin{aligned} & Employ(E3) \quad \sqcap \quad Use(U2) \quad \sqcap \\ & lcs(\delta Permission(P1), \delta Permission(P2)) \\ \equiv & \quad Employ(E3) \quad \sqcap \quad Use(U2) \quad \sqcap \\ & lcs(\delta Permission(P1), \delta Permission(P1)^\epsilon) \end{aligned}$$

using lcs properties, we obtain:

$$\equiv Employ(E3) \sqcap Use(U2) \sqcap \delta Permission(P1)$$

Using security rules, we can deduce that the precedent proposition subsumes $\delta Is - permitted(I1)$.

And, because $Is - permitted(I1) \sqsubseteq \delta Is - permitted(I1)$, we can deduce $Is-permitted(I1)$. Therefore Jean is permitted to read SP1 when one of these contexts is true (absence of assistant, absence of assistant with substitute present).

VII. CONCLUSION AND FUTURE WORK

The work presented in this paper has led to the definition of a new system based on description logic that is expressive enough to be used as part of an application and to represent default knowledge and exceptional knowledge. The $JClassic_{\delta\epsilon}^+$ highlights the interests and the relevance of defaults in conceptual definition. For the $JClassic_{\delta\epsilon}^+$ language, we have given a set of axioms outlining the essential properties of the connectives from this definitional point of view: property links default characteristics to exceptional or strict ones. This set of

axioms induces a class of $CL_{\delta\epsilon}^+$ -algebra of which the terms are concept descriptions. Using the conjunction connectives \sqcap and "lcs", the set of concept can be partially ordered w.r.t the equational system (descriptive subsumption in free algebra). $JClassic_{\delta\epsilon}^+$ is defined with a universal algebraic corresponding to a denotational semantic, where terms are denoted exactly by sets of strict and default properties.

This system consists of three modules: a module for representing knowledge, a module to use that knowledge and a module to update knowledge. The module which allows to use knowledge is endowed with a subsumption algorithm which is correct, complete and polynomial.

In our work, the description logic is endowed with an algebraic intensional semantics, in which concepts are denoted by a normal form of all their properties. These normal forms (i.e., elements of the intensional semantic) are used directly as an input to the algorithm of subsumption and algorithm of deductive inferences.

The developed tool has been used to describe our contextual access control model in which authorization is assigned to a subject according to its role in an organization in a given context. The two operators of default and exception are used in a clever way to assign permission depending on the context. Each time, the context changes, permissions are redefined and re-assigned to subjects.

Context can take several values, it can be a default one, an exception to the actual context, conjunction of contexts or disjunction of context. In this paper, we specially lay emphasis upon the last one.

An interesting topic for future research is to extend our tool to take into account spacial-temporal context to make our system more expressive with keeping a reasonable complexity. We also envisage to explore other appropriate and real applications.

REFERENCES

- [1] N. Boustia and A. Mokhtari. $JClassic_{\delta\epsilon}^+$: A Description Logic Reasoning Tool: Application to Dynamic Access Control. In *Proc. The Second International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, Computation Tools'11*, September 25-30, 2011, Rome, pp. 25-30, ISBN: 978-1-61208-159-5.
- [2] D. E. Denning. Multilevel secure database systems: Requirements and model. In *NAS/AFSB Summer Study on Multilevel Database Management Security, Working Paper*, June 1982.
- [3] F. Coupey and C. Fouqueré. Extending conceptual definitions with default knowledge. *Computational Intelligence*, vol 13, no 2, pp. 258-299, 1997.
- [4] <http://www.cs.man.ac.uk/~sattler/reasoners.html>. April, 2012.
- [5] F.M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Principles of language representation and reasoning: second international conference*, pp. 151-162, 1991.
- [6] N. Boustia and A. Mokhtari. A Contextual Multilevel Access Control Model. In *Int. J. Internet Technology and Secured Transactions*, Vol. 3, No. 4, pp. 354-372, 2011.
- [7] N. Boustia and A. Mokhtari. A dynamic access control model. In *Applied Intelligence Journal*, Volume 36, Number 1, pp. 190-207, DOI 10.1007/s10489-010-0254-z, 2012.
- [8] A. Borgida and P.F. Patel-Schneider. A Semantic and Complete algorithm for subsumption in the CLASSIC description logic. *Artificial Intelligence Research*, vol 1, pp. 277-308, 1994.

- [9] W.W. Cohen, A. Borgida and H. Hirsh. Computing Least Common Subsumer in Description Logic. In *10th National Conference of the American Association for Artificial Intelligence*, pp. 754-760, San Jose, California, 1992.
- [10] V. Ventos and P. Brésellec. Least Common Subsumption as a connective. In *Proceeding of International Workshop on Description Logic*, Paris, France, 1997.
- [11] V. Ventos, P. Brésellec and H. Soldano. Explicitly Using Default Knowledge in Concept Learning: An Extended Description Logics Plus Strict and Default Rules. In *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001*, pp. 173-185, Vienna, Austria, September 17-19, 2001.
- [12] A. Corradi, R. Montanari, and D. Tibaldi. Context-Based Access Control in Ubiquitous Environments. In *3rd IEEE International Symposium on Network Computing and Applications (NCA)*, August 2004.
- [13] S. Fu and C.-Z. Xu. A Coordinated Spatio-Temporal Access Control Model for Mobile Computing in Coalition Environments. In *In 19th IEEE International Parallel and Distributed Processing Symposium*, April 2005.
- [14] F. Hansen and V. Oleshchuk. Spatial Role-Based Access Control Model for Wireless Networks. In *IEEE 58th Vehicular Technology Conference, VTC 2003-Fall*, volume 3, October 2003.
- [15] M. Strembeck and G. Neumann. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environements. In *ACM transactions on information and System Security*, 7(3), pp.392-427, 2004.
- [16] H. Wedde and M. Lischka. Role-Based Access Control in Ambient and Remote Space. In *9th ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, USA, June 2004.
- [17] G. Zhang and M. Parashar. Dynamic Context-aware Access Control for Grid Applications. In *4th International Workshop on Grid Computing*, November 2003.
- [18] J.B.D. Joshi, E. Bertino, and A. Ghafoor. Generalized Temporal Role-Based Access Control Model. 17(1), pp. 4-23, January 2005.
- [19] C.K. Georgiadis, I. Mavridis, G. Pangalos and R.K. Thomas. Flexible Team-based Access Control Using Contexts. In *Sixth ACM Symposium on Access Control Models and Technologies*, pp. 21-27. ACM Press, Chantilly, 2001.
- [20] J. Hu and A.C. Weaver. A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications. In *First Workshop on Pervasive Privacy Security, Privacy, and Trust*, Boston, MA, USA, 2004, <http://www.pspt.org/techprog.html>
- [21] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, Lake Come, Italie, June 2003.
- [22] F. Cuppens and A. Miège. Modelling Contexts in the ORBAC Model. In *19th Annual Computer Security Applications Conference*, Las Vegas, December 2003.
- [23] H. Debar, Y. Thomas, F. Cuppens and N. Cuppens-Boulahia. Enabling Automated Threat Response through the Use of a Dynamic Security Policy In *Journal in Computer Virology (JCV)*, 3, 3 (2007), pp. 195-210, 2007.
- [24] N. Boustia and A. Mokhtari. Representation and reasoning on ORBAC: Description Logic with Defaults and Exceptions Approach. In *Workshop on Privacy and Security - Artificial Intelligence (PSAI)*, pp. 1008-1012, ARES'08, Spain, 2008.
- [25] N. Boustia and A. Mokhtari. $DL_{\delta\epsilon} - OrBAC$: Context based Access Control. In *Proc. WOSIS'09*, pp. 111-118, Italy, 2009.
- [26] R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, L. Alperin Resnick, and A. Borgida. Living with CLASSIC: When and How to Use a KL-ONE-Like Language. In *John Sowa, ed., Principles of Semantic Networks: Explorations in the representation of knowledge*, pp. 401-456, Morgan-Kaufmann: San Mateo, California, 1991.
- [27] R.J. Brachman, D.L. McGuinness, L. Alperin Resnick, and A. Borgida. CLASSIC: A Structural Data Model for Objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pp. 59-67, June 1989.
- [28] D.E. Bell, and L.J. LaPadula. Secure Computer System: Unified Exposition and Multics Interpretation. In *MITRE Technical Report MTR-2997*, July, 1975.

Seeing the Big Picture: Influence of Global Factors on Local Decisions

Terry Bossomaier*, Michael Harré*[†], Vaenthan Thiruvardchelvan*

*Centre for Research in Complex Systems

Charles Sturt University

Bathurst, Australia

e-mail: {tbossomaier,vthiru}@csu.edu.au

[†]Centre for the Mind

University of Sydney

Sydney, Australia

e-mail: michael.harre@sydney.edu.au

Abstract—This paper extends recent work studying the development of human expertise in the game of Go. Although it appears like a simple game on the surface, Go is actually the most difficult of all established games for artificial intelligence, with no computer program yet reaching the top international level on a full 19×19 board. On smaller boards with sizes like 9×9, computers are competitive, implying that the understanding of complex global interactions is the key to human superiority. The temporal analysis of game positions yields some interesting insights into local/global analysis. By mining thousands of positions from online games, we show that at some player levels, the sequence of plays leading up to a local position is a stronger determinant of the next move than the position alone. This suggests that the sequence of plays is an indicator of global strategic factors and thus provides a context for the next move in addition to the local position. Using perceptual templates introduced in other work, we demonstrate that this global context appears at the very earliest stages of cognition.

Keywords—game of go; decision making; entropy; online data mining

I. INTRODUCTION

The big picture often influences or overrides local factors in many areas of human expertise, from board games to politics. Challenging games, such as Chess and Go, provide an excellent framework for studying expertise [1][2][3][4], since they are both strategically deep but tightly constrained. This paper presents a striking demonstration of this, using data mined from thousands of decisions in online games. In recent work, we have demonstrated transitions in the acquisition of expertise in the game of Go [5]. This game is interesting because it is currently the most difficult of all established games for computational intelligence. This contrasts with Chess, where the IBM computer Deep blue [6][7] was able to defeat world champion Garry Kasparov.

We also demonstrated therein, from calculation of mutual information (eqn. 5) between moves, that one of these has the character of a *phase transition* [8]. The idea of a phase transition comes originally from physics, from the study of phenomena like the melting of ice to give water. When

such a physical phase transition occurs, there is a dramatic reorganisation of the system. In this case, water molecules which were fixed rigidly in place in ice become free to move around, and perhaps travel long distances. During a phase transition, systems exhibit long-range order, where there are correlations in activity or structure over large distances and system parameters often exhibit power-law behaviour, or fat-tailed distributions. Another example of a phase transition is in adding edges to random graphs. At a certain point each graph shows a transition: the average path length (the number of steps from one node to another) rises to a peak, and then drops back down again.

A dynamical system example is the Vicsek model developed for studying magnetic transitions in solid-state physics [9]. In this model particles travel around a two dimensional grid, and when they come within some specified distance of each other, their directions of movement partially align. Phase transitions occur in this system as particles flow around in groups, like flocks of birds, but dynamically—continually forming and dissolving.

Mutual information is a precisely defined quantity, originating from Shannon's mathematical theory of communication [10]. It is a system property which measures the extent to which the structure or behaviour of one part of a system predicts the behaviour of another. In the Vicsek model above, the direction and velocity of one particle provides some information about the direction of all the other particles. The mutual information peaks during the phase transition [9][11] and, along with other characteristics like long-range order and power-law behaviour, is thought to be a general property of phase transitions.

Previous work [8] has already demonstrated phase transitions in collective human decisions in Go. In this paper, we found a peak in mutual information as a function of rank amongst Go players, from 1 Dan Amateur through to the very top players, 9 Dan Professionals [8]. We also present evidence that there is global influence on local decisions, and that the influence is greatest during the phase transition. The evidence

for the global factors arises from temporal analysis: the next move is more predictable given the sequence which led up to it, compared with just using the position at which it is made. We argue below that this arises from the global information inferred from the sequence.

Section II describes the conceptual background or expertise (sections II-B and III-B discuss *perceptual templates*, which form one of the earliest stages of processing of a Go board. It turns out that for professional players, these templates have a strong non-local character, supporting the findings from mutual information.) Sections III and IV describe the methods and results respectively. The discussion (section V) and conclusions (section VI) round off the paper.

II. EXPERTISE AND PERCEPTION

The study of expertise in games owes much to Fernand Gobet and his colleagues, summarised in his book *Moves in Mind* [1]. However, the methods used in Go in this paper rely on a new methodology introduced in [12]. The next two sections discuss these in turn.

A. State of the Art in Game Expertise

Much of the work on human expertise has been based on games, especially Chess, as in Gobet's extensive work [1][13]. One of the key ideas, essentially from Nobel Laureate Herbert Simon, is that human expertise involves building a huge library of patterns [14][15]. The application of these ideas in artificial intelligence for games is relatively new however [16].

These patterns build up through the formation of *chunks*, psychological observables like the memory of Chess positions, well predicted by models like CHREST [3]. The way the cognitive structures in the brain might change as expertise develops, and in particular the appearance of phase transitions, is a relatively new idea introduced by Harré and Bosso-maier [8][5].

Further recent advances have been limited, particularly in Go, where a combination of the gamespace complexity [17] and a lack of genuinely human-like heuristics like an evaluation function make progress difficult. However with the development of ever more effective random sampling techniques, such as the UCT-Monte Carlo approach currently favoured by AI system developers [18], some progress has been made in achieving strong amateur play. However, these techniques do not address the inherent complexity of the game nor the techniques that humans have developed in order to address this, almost completely because it is difficult to investigate.

Of relevance to game players is the current state of the game, the likely future states of the game and in what order those future moves will be played. The current state of the game is very well approximated by the pieces currently on the board (this excludes some technical rules about repeated positions that are only rarely relevant), and these can be divided up loosely into tactical, strategic and distracting pieces. Tactical pieces are involved in local battles for territory, while strategic pieces play a role in long-term plans spanning the entire board. Distractors play only weak roles in either of the

preceding plan types. Of course, a single stone can participate in both local and global strategies. In terms of future states of the game, we considered only local patterns and what was played in the local area – a purely tactical aspect of the game. This leaves only the strategic relationships as a source of information that might perturb the actual moves made. It is this external influence on tactical plays that is implicit in the global contextual analysis of this paper.

We argue that the sources of information players use in order to make good decisions are of two types: *local* and *global*. Every level of player in our study has learned a great deal about the game of Go over the course of their lives; we now want to make explicit and quantify this information. We do this by looking at the probability distributions of moves made in a variety of different positions. The relevance of the division of the problem space into these two parts can be seen in the work of Stern et al. [19]. They were able to produce 'best-in-class' move prediction for professional players in Go, achieving a 34% success rate. This was achieved by training their system on 181,000 expert game records and using a Bayesian framework for matching moves to positions.

The level of success achieved in this work highlights one of the principal difficulties of good performance in complex tasks: exact pattern matching is not enough. AI systems need to be able to model how non-local aspects – i.e., information that cannot be derived by exactly matching board configurations – influence decisions. Loosely interpreted, this is what is called *influence* in Go and had not been reported in the research literature before our recent work.

B. Kohonen Maps & Perceptual Templates

If local decisions involve global factors, the question arises as to where in the cognitive hierarchy global information appears. We use the recent work on *perceptual templates* to show that it starts at the very lowest levels. Perceptual templates are the building blocks of perception, experienced preattentively and fundamental to rapid decision making – the instant appraisal of situations by experts, the guiding of eye movements and expert memory for real-world positions.

A novel way to determine such perceptual templates involves the use of Kohonen maps trained on game data [12]. The templates so found can then be analysed for global properties. Teuvo Kohonen [20] introduced self-organising maps (SOMs) as a model of human visual information processing. Although they help explain some structural characteristics of the visual cortex, they have found considerable practical use in the signal-processing domain, especially image processing.

A SOM is a competitive learning process, comprising of a selectable number of neurons. Each neuron has a random weight vector, and a set of inputs of the same dimension. In the case of an image, the inputs would be the colour components of each pixel.

Training proceeds as follows. A pattern from the training set is presented to each neuron in the map. There will be one neuron which is closer (different metrics of proximity may be used) to the pattern than any of the others. The weights of

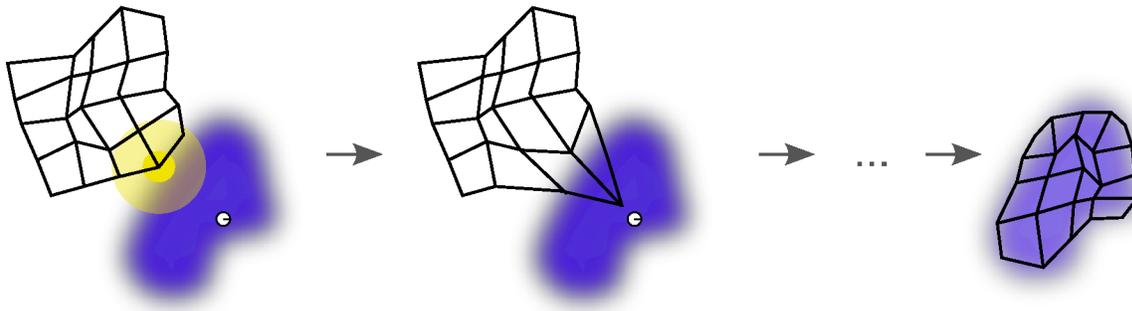


Figure 1. This illustration captures the essence of the Kohonen map or Self-Organising Map (SOM), mapping two dimensions to two. The blue cloud represents the training dataset, and the intersections of the grid represent the weight vectors of the neurons in the map. Initially the weights are random and the map does not correlate with the data (left). Training the map involves finding the neuron with the closest weight vector (yellow dot) for each datapoint (white dot), and adjusting the weight towards it (center). After repeated training, the map comes to approximate the dataset (right). In our work, the dataset and weight vectors are 361-dimensional (intersections on the Go board), while the map is 2D with 50x50 neurons. Copyright Mclrd [CC-BY-SA-3.0 or GFDL], via Wikimedia Commons. <http://commons.wikimedia.org/wiki/File:Somtraining.svg>

this and neighboring neurons are then increased while neurons further away are decreased. Presentation of patterns is repeated until the weights converge. Fig. 1 illustrates this process.

The competitive, winner-takes-all strategy is difficult to analyse theoretically. There are some results on the proportion of neurons devoted to the probability density of particular features, but such results are often derived under simplifying assumptions. Empirically what happens is that the neurons end up representing commonly occurring sub-features. Unlike multi-layer feedforward networks, the basic SOM has no capacity for translational or geometric transformation. In image processing this is often a disadvantage, but with the fixed dimensions of the Go board, it is actually advantageous here.

We have used two different techniques in this work that have quite separate roles and interpretations in making decisions in any complex environment. The perceptual templates are a learned, static and sparse representation of the game state that is implemented as a perceptual guide to inform higher order cognitive processes. One type of high-order cognitive processing is exact pattern matching and planning, and it is this aspect that is studied using information theory. These two aspects are necessarily disjoint as they are plausible subsystems that are likely present in Kahneman's 'Dual Processing' account of human cognition [21]. In this interpretation, perceptual processes are fast to implement (as they are parallel processes), slow to learn and unlimited in their capacity. In contrast, planning and evaluation processes are most likely slow to implement (as they are serial processes) but are deployed dynamically and have limited capacity.

III. METHODS

Harré and Bossomaier [8] examined game trees six moves deep (i.e., three black and three white) for around 8,000 games across a range of Go expertise. At the low end were 2 Kyu Amateurs, a rank reached by serious players after a couple of years club play, through the highest amateur rank of 6 Dan Amateur (6A), to the top professional rank of 9 Dan

Professional (9P). Game data was obtained from the *Pandanet* Go server. Full details of experimental procedures are given in Harré et al. [8]. The game trees were computed from 7x7 board sections in the corners, from games played between players of the same rank. No symmetry was exploited, apart from rotations to align each of the four corners (used to maximise data yield per game). Note that although these are the first six moves played in the region, they are not necessarily the first six moves of the game.

A. Information Theory

Information theory was introduced by Shannon [10] to study the communication of data through channels. It has since proved immensely useful across many domains. It is essentially a statistical concept, quantifying the gains and losses from choices amongst alternatives.

Shannon defined the information of an event in bits, η_x as:

$$\eta_x = -\log_2 P(x) \quad (1)$$

where $P(x)$ is the probability of observing the event x . If we now average over the set M , of all events which could occur, denoted by x_i , we get the average information, which is also known as the *entropy*:

$$H(M) = -\sum_i P(x_i) \log_2 [P(x_i)] \quad (2)$$

Applying this to moves in Go gives us a measure of how likely one move is over another. If the entropy is zero, then $P(x_i)$ must be one for some x_i and zero for all others. Thus the move is completely determined without ambiguity. Entropy is a measure of disorder or randomness and is maximal when the probabilities for all events are the same.

We need one additional concept, the *conditional entropy*. The likelihood of any given move is of course dependent upon previous moves up to that point. Thus we can ask what the probability of a move is in a given position. From this, we can ask what the entropy is in the set of moves if we know

the position in which the move was made. If we now average over all positions, we get the entropy of moves given positions. This is the conditional entropy, defined in eqn. 4.

For each possible move on the Go board m_i , three probability distributions were computed:

- 1) the probability of the move occurring, $P(m_i)$
- 2) the conditional probability, $P(m_i|q_i)$, of the move, m_i occurring from a given position, q_i
- 3) the conditional probability, $P(m_i|s_i)$ of the move occurring from a given position, *reached by a particular order of moves*, s_i .

From these results, the entropy and mutual information (eqn. 5) were calculated, but this paper addresses findings from the entropies alone. A discussion of the primary results from mutual information is given in [8].

The move entropy, $H(M)$, is taken over all moves which can arise at each level in the game tree (i.e., for the six moves in the sequence):

$$H(M) = - \sum_i P(m_i) \log_2[P(m_i)] \quad (3)$$

For the first move in the region there are 49 possible positions, decreasing to 44 after five moves, giving a maximal entropy of $\log_2 44 = 5.5$ bits, which would occur if all moves were equally likely. But since the moves are chosen strategically, they are far from random, so the measured entropies are much lower than this.

The conditional entropy, $C(M|Q)$, is the move entropy calculated from the moves which can arise in a given context, such as position q_j , or sequence of moves s_j leading to a position:

$$C(M|Q) = - \sum_i \sum_j P(m_i|q_j) \log_2[P(m_i|q_j)] \quad (4)$$

From the conditional entropies, we can calculate the mutual information, $I(Q, M)$ using Shannon's formula [10], eqn. 5:

$$I(Q, M) = H(M) - C(M|Q) \quad (5)$$

The same expressions are used for an ordered sequence of moves, replacing q_j with s_j . These entropic quantities are now calculated across all ranks, from 2 Kyu Amateur (*am2kyu*), through the amateur ranks to *am6d*, onto the highest rank of 9 Dan Professional (*pr9d*). The results are shown in Figs. 1–3.

B. Perceptual Templates

For our work, we use a separate 50x50 SOM (2500 neurons) for each of the 361 intersections of the Go board. Each SOM is trained on board states directly preceding a move at that point, mined from the online gameplay database. Board states are represented as linearized length-361 vectors with values equal to -1 , 0 or 1 , representing black stone, empty or white stone respectively. Games were normalized to always start with a white stone, and no deduplication along axes of symmetry

Threshold	# Templates	Average Size
0.9	10,929	11.1
0.8	26,318	13.8
0.7	55,553	15.2
0.6	145,534	16.5
0.5	364,557	18.3

TABLE I. Number of perceptual templates and their average sizes (maximum Euclidean distances) per threshold.

was carried out. The weight vectors of the neurons was also constrained to this range, facilitating easy template extraction.

Further details may be found in [12], from which trained SOMs were reused. The ones used for this paper are taken from games 5 Dan Professional and above. 18,000 games were used in training each map.

The spatial topology of a trained SOM is usually of significance in typical uses, however we discard this information. We consider the weight vectors of each neuron at every point as potential perceptual templates. Neurons which have strongly learned patterns of stones across the board will then be extracted as templates. Thus the number of potential templates equals the total number of neurons, $361 \times 2500 = 902,500$.

However, since there are on average only about 7 training games per neuron, the learned weights are still quite noisy. Therefore, actual templates are extracted from the weight vectors by thresholding the weights to 1 , 0 , or -1 using thresholds of k and $-k$, for values of k of 0.5 , 0.6 , 0.7 , 0.8 and 0.9 . After thresholding, empty and duplicate templates are removed, leaving useful templates. Table I records the resulting number of templates at each threshold. Examples of these templates are shown in Fig. 2, which may be locally clustered as in subfigures a. and b., but are often non-local as in c. and d. Further procedural details can be found in [12].

IV. RESULTS

Fig. 3 summarises the key findings of the paper. It shows the conditional entropy as a function of move in the sequence of six, averaged across all ranks, both amateur and professional. Note that the moves are logged as they occur in the game. They are *not* necessarily in sequence. In other words, this is not a game on a small board region but a window on a full 19×19 game. Since the standard of play is professional for this analysis, extremely weak moves are unlikely to occur and will not appear in the game records. Error bars are calculated as in Harré et al. [5]. Up to move three, the entropy for both the ordered and unordered cases are the same. At move three, they fall dramatically, but the ordered average falls about a third more.

Fig. 4 shows the entropy at each move from a given position. For purely random moves, the entropy at each move in the sequence would be between 5 and 6 bits (Section III). The entropies observed are of course much lower – usually less than 2 bits – reflecting the structure inherent in the game.

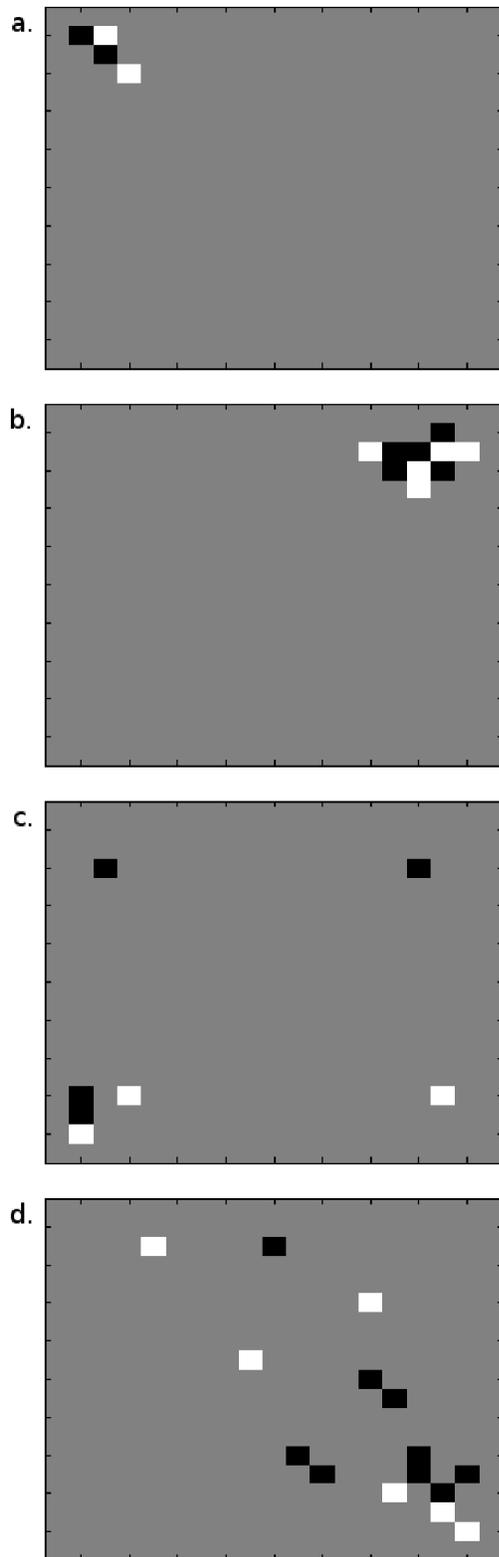


Figure 2. Four templates arising from a threshold of 0.9. Most templates are clustered in a corner of the board like a. and b. Template c. spans multiple corners, as games commonly develop. Template d. reflects a game which spreads outwards rather than clustering in the corners.

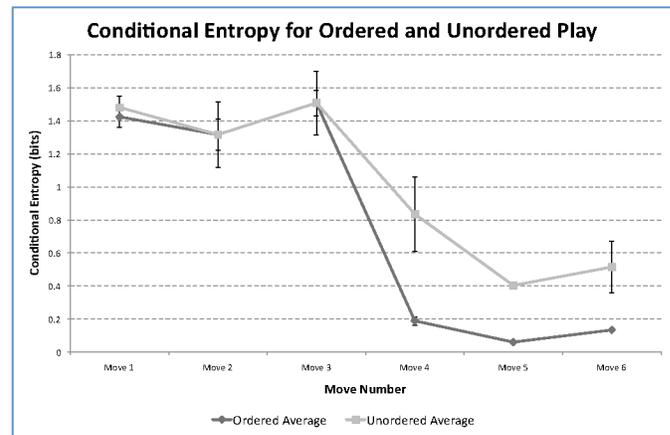


Figure 3. Conditional entropy (eqn. 4) as a function of move averaged over all ranks

The entropy for the third move is on the order of the first two, but the entropy falls a little for the fourth move and a lot more for the fifth and sixth moves. This is not surprising given the reduced options available as the number of stones on the board increases. The entropy summed over all moves declines linearly to the maximum amateur rank and then increases gradually from from the first professional rank onwards.

Fig. 5 shows the entropies resulting from positions which arose from a particular sequence of play. These entropies are around 3 bits smaller than the unordered case. The slope of the regression line for the amateur levels is not so large, but the trend for the professionals displays a different pattern: the summed entropy jumps near the start of the professional ranks and then *decreases* with rank up to 9P, with a slope very similar to that for the amateur ranks on the left of the figure.

The most interesting thing about this figure, though, is the way the entropy for the last three moves shrinks and vanishes as the amateur rank increases from 4A to 6A. In fact the summed entropy for the first three moves is quite similar to the unordered case, so the three bit loss is almost all in the last three moves.

Turning to the templates derived from the SOMs, as the threshold is reduced from 0.9 to 0.5 (Figs. 6–10) the number of templates increases exponentially, but remarkably, the shapes of the histograms do not change so rapidly. So the number of stones in a template is around 5 for the bulk of them, for thresholds 0.6 and upwards. As the threshold drops, the new templates which appear are new patterns of a small number of stones, rather than patterns with increasing numbers of stones. (The last bar includes all templates with higher stone counts).

Table I includes the averages of the maximum Euclidean distance between two stones in each template, per threshold. We see that it rises fairly linearly with threshold. Figs. 11 – 15 are histograms of those distances.

The theoretical maximum distance is 27, but since this includes stones on opposing corners, the maximum observed distance will usually be lower. At a threshold of 0.9, over half

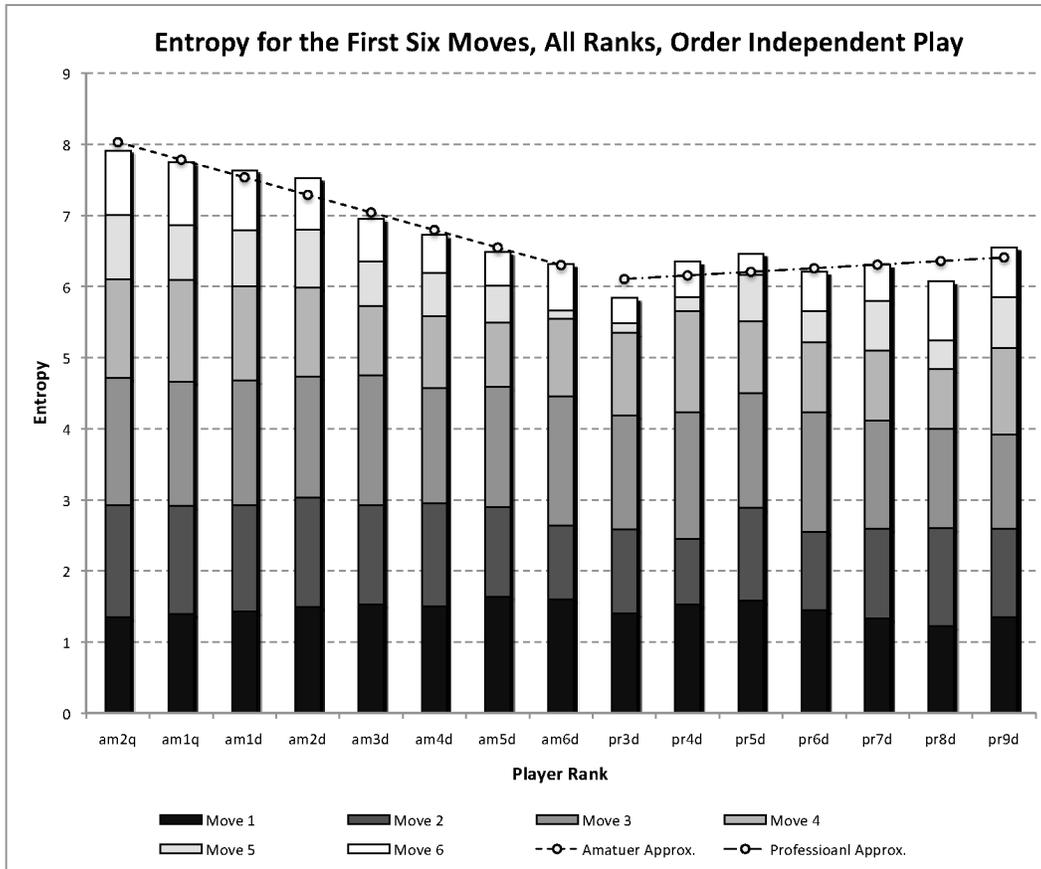


Figure 4. Entropies for moves from a given board position (reprinted from [8])

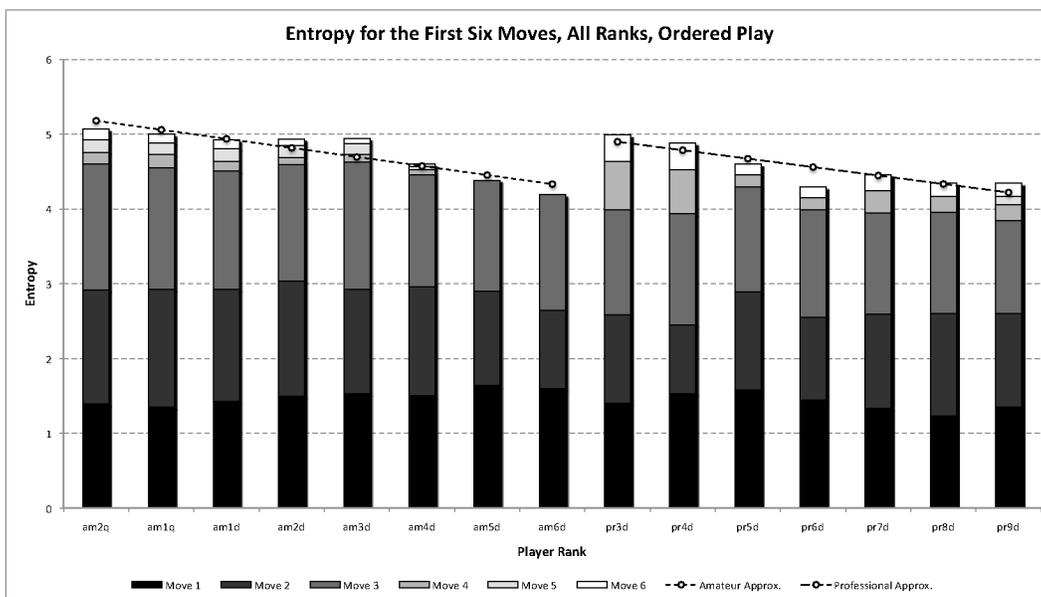


Figure 5. Entropy for the first six moves shown as a stacked bar chart. The black bars represent the entropy at move 1, the dark grey at move 2 and so on for all six moves. The dashed regression lines show the total entropy for the amateur and professional sequences.

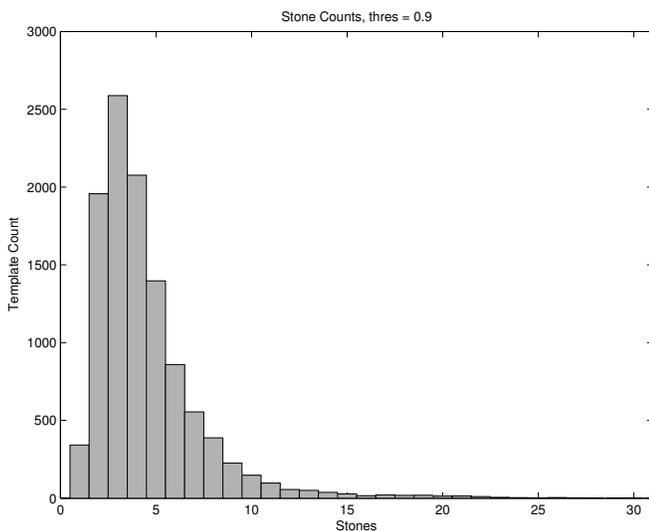


Figure 6. Number of templates containing a given number of stones, threshold = 0.9.

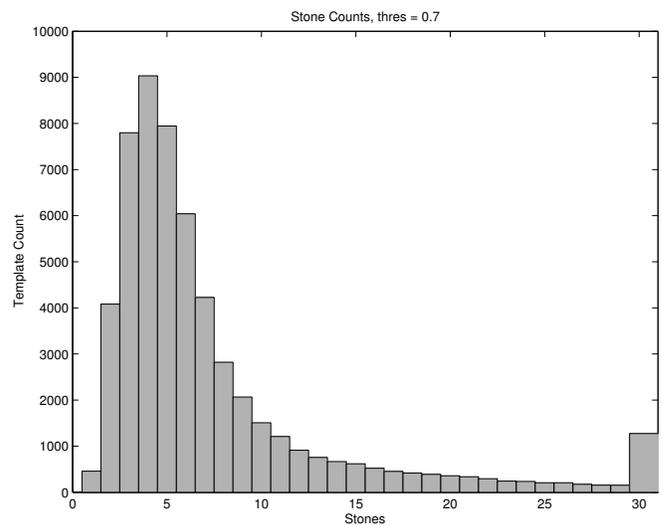


Figure 8. Number of templates containing a given number of stones, threshold = 0.7.

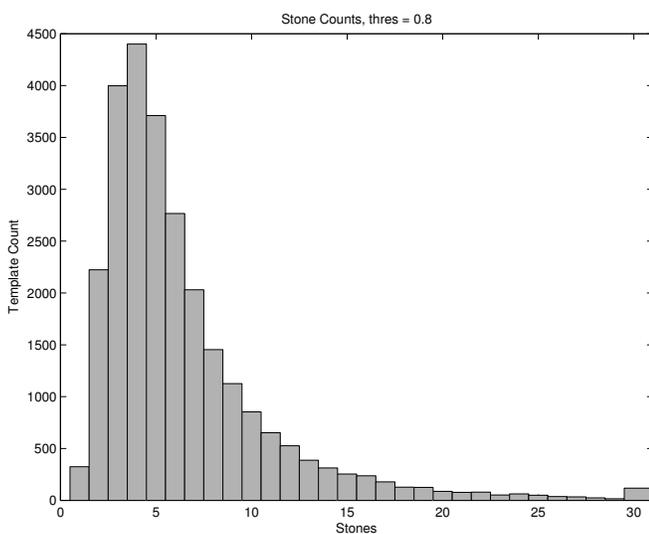


Figure 7. Number of templates containing a given number of stones, threshold = 0.8.

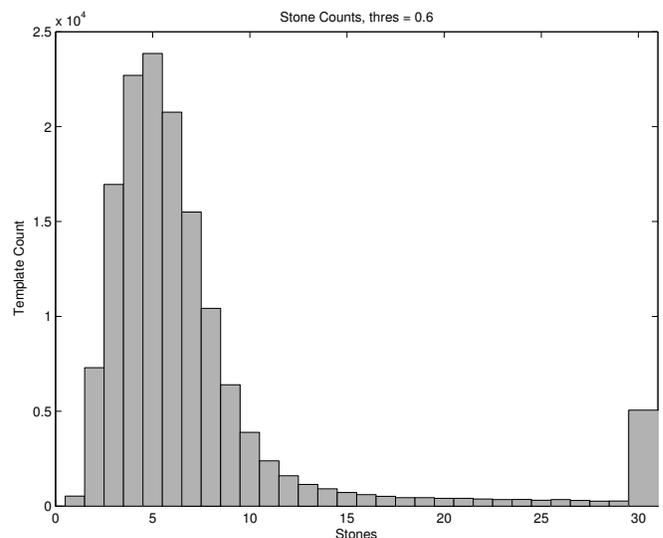


Figure 9. Number of templates containing a given number of stones, threshold = 0.6.

the templates have a distance of 12 or greater, implying that they include stones in different corners of the board. There could be a small number of templates occupying one corner and the centre, but these would need to have distances around 12, where we find a minima instead.

Also salient are the twin peaks around 3 and 18, possibly representing corner clusters and cross-board patterns respectively. As we lower the threshold, the proportion of templates spanning the board increases until it dominates at 0.5, while the peak at 3 stones vanishes.

V. DISCUSSION

There are three very interesting features of these results, which we consider in turn: a) the difference between ordered and unordered play, b) the way the conditional entropy varies with rank, and c) how perceptual templates span the entire

board.

That the ordered and unordered play differ, implies that the position at each move is *not* the sole determinant of the opponent response. The much lower conditional entropy after the first three moves for the ordered case strongly suggests that the sequence of moves has revealed something of the global context which has in turn fed back into move selection. To see this, imagine that black is strong in one area of the board and white in another. Since relationships between localised groups of stones are of great strategic importance in Go, the locations of these areas will strongly influence the order of moves made in the local area we examine. The first three moves implicitly contain some of this information, which subsequently reduces the range of options in the next three moves.

The gradual decline in entropy with rank for amateur and professional reflects a gradual reduction in the space or range

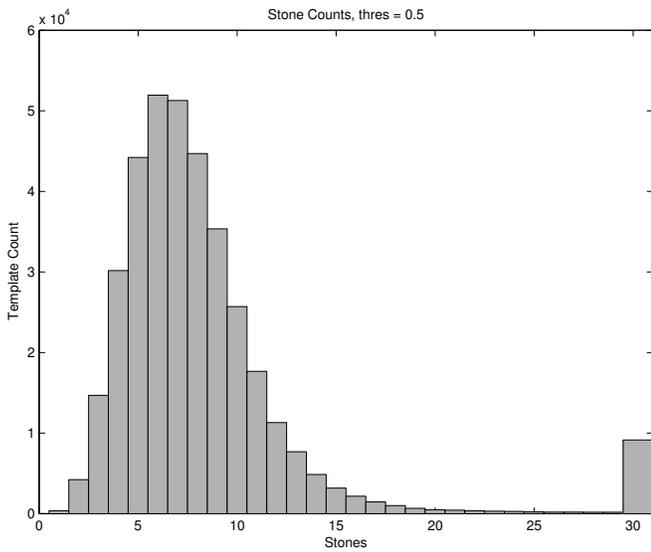


Figure 10. Number of templates containing a given number of stones, threshold = 0.5.

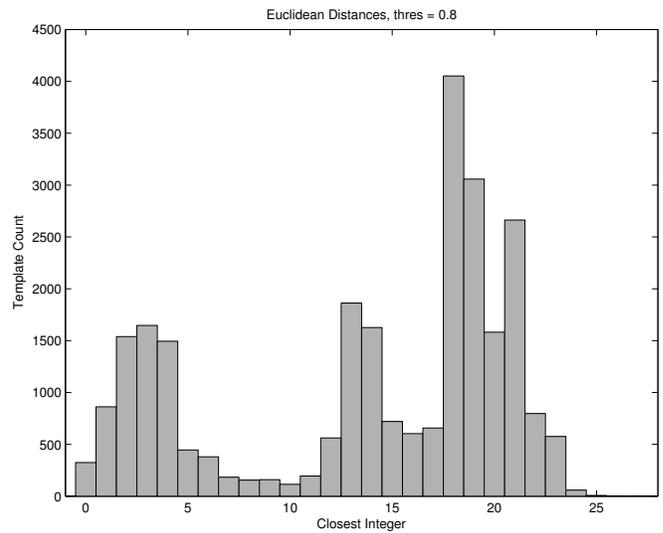


Figure 12. Maximum Euclidean distance between stones in a template, threshold = 0.8.

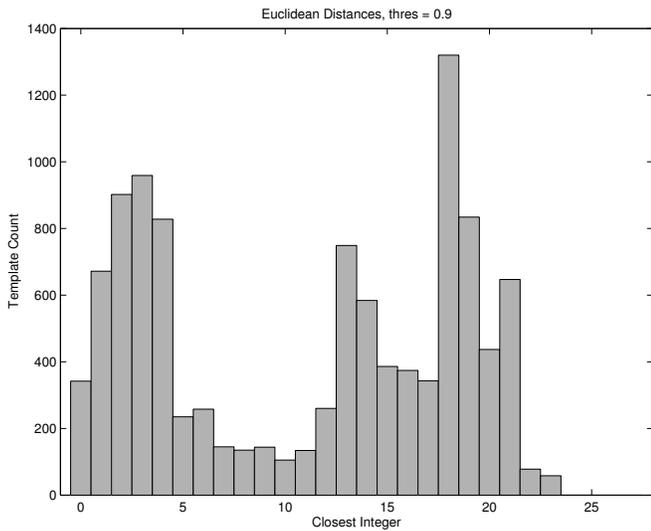


Figure 11. Maximum Euclidean distance between stones in a template, threshold = 0.9.

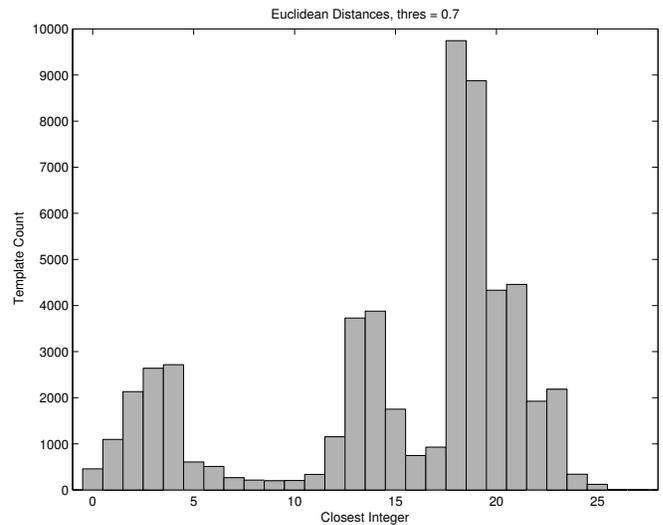


Figure 13. Maximum Euclidean distance between stones in a template, threshold = 0.7.

of options, which we could conceive of as the elimination of poor moves in established situations, similar to mastering the openings in Chess.

Our data and results are explicitly based on an analysis of the local information, but by implication they also say a great deal about the global context that influences these localised decisions. The first three moves in our study have a reasonably similar conditional entropy of about 1.4 – 1.6 bits of information. This is the amount of information that is common between each successive move within the local region. Such measures of information are the best estimate of how much one stochastic variable can tell us about another [10].

The only other source of information available to the players are the pieces on the board that were not included within our local region. We exclude the possibility of being able

to read the other opponent. While it is a debated issue as to the importance of opponent-reading skills in a complete information game such as Go, we believe that it is relatively insignificant. The strategical influence of the other stones on the board that were not within the local area of study, is a much more significant factor. The changing influence that non-local information has on decisions during a game, is evident in the significant drop-off in the conditional entropy after move three in Fig. 3, a drop in shared information of nearly an order of magnitude for the ordered play and about half that for unordered play. This is consistent with the observation that, at the time of writing, the best computer Go programs are close to professional-level on small boards like 7×7, but rapidly deteriorate on larger boards, as global influences become important.

This change in conditional entropy in the corner regions of

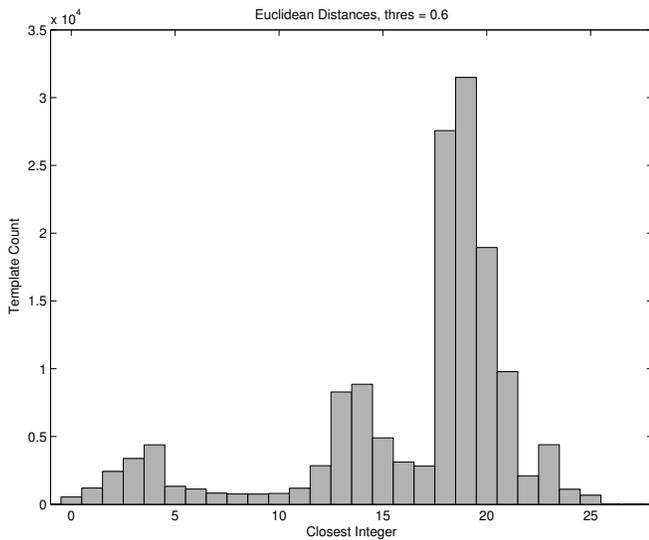


Figure 14. Maximum Euclidean distance between stones in a template, threshold = 0.6.

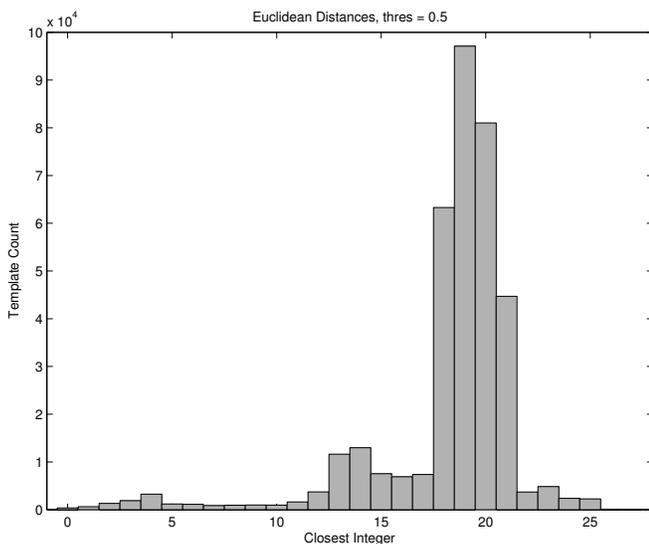


Figure 15. Maximum Euclidean distance between stones in a template, threshold = 0.5.

the board as the game progresses might be due to the shrinking size of the move space as the board fills up. While this might have some minor influence on our results, we should also expect such changes to be almost linear as the number of available positions only drops by a total of $1/43$ per move. It is also possible, but exceptionally unlikely, that after the third move, players choose much more randomly – i.e., without concern for pieces on the board, local or not – than they did for the first three moves. Considering the vast training literature available to players that readily teaches them the many different variations of the first six moves within a corner, and how to contextualise these decisions by considering what stones occupy nearby areas, we consider this to be an unlikely proposition.

Instead we argue that it is just this external influence, the

influence of the stones arrayed on the rest of the board that is having such a striking influence on the conditional entropy. This is perhaps not so surprising when considered in the light of the state of the game itself, after three moves have been played in the corner. These first moves can be thought of as establishing the board layout in terms of an ‘opening book’; highly stylised placement of local stones, where the local pattern can be thought of as effectively uncoupled from the rest of the board, or at least equally coupled for these first moves. This coupling then changes significantly from the fourth move onwards, where greater consideration needs to be afforded to the other pieces on the board. This change in the focus of gameplay significantly reduces the information coupling between local moves and local stones on the board.

The use of global information is supported by the analysis of the perceptual templates. A large fraction of templates cover more than one corner of the board, implying that global analysis *starts at the very earliest perceptual levels*. As the threshold is reduced to 0.5, the number of local templates actually gets eclipsed by non-local ones. An additional finding from our results is that the majority of perceptual templates contain less than ten stones, regardless of noise threshold and number of templates. Even at the 0.5 threshold, where most templates are non-local, the average number of stones remains small. This median figure of around 5 – 7 is on the order of human working memory capacity and similar to the figures in Gobet’s CHREST models [1]. This is therefore an important issue, subject to future research.

The complete disappearance of entropy at the high amateur ranks is very interesting. It suggests that at this level, play has become somewhat stereotyped, and a major change in thinking is needed to advance—which indeed seems to happen on turning professional. Thus, this loss of entropy is consistent with the long-range order found in phase transitions by Harré et al. [8]. They observed a peak in mutual information at the transition to professional play, indicating some sort of major cognitive reorganisation.

At present, we do not know how to quantify such a reorganisation, and this remains an exciting open question. Ongoing work is attempting to apply the CHREST models to Go [3], and to determine how the phase transitions might be predicted.

A. Implications for Computer Go

The objective of this study was to determine some characteristics of human Go expertise. These may subsequently be fed into the computer Go domain, but that was not our motivation here. Our analysis is once-off, so the time-complexity of our computations is irrelevant. The methods as described in this paper have never been used before, and no prior work has attempted to identify the influence of global factors in Go.

VI. CONCLUSIONS

The analysis of large volumes of data has generated powerful new insights into human cognition in the Game of Go, with potential applicability to other domains. We have shown

that low-level perceptual templates of professional players are non-local, i.e., include features from the whole board. The paper links this to earlier work on mutual information in local positions, which we infer to be influenced by global factors. The *sequence* of moves leading to a position was shown to provide more information about the next move than the position alone, which could be accounted for by global contextual information provided by the former.

The big challenge for future work is to determine if these properties hold in other domains. Poker is the ideal next game to study: it is the second most difficult established game for computers to play well, and has the additional features of incomplete information, stochastic elements and theory of mind.

In 2012, Zen, one of the top computer Go programs, won games against 6 Dan Amateur players, and came much closer to beating professional-level players than any program has before. But computer Go relies heavily on Monte Carlo Tree Search, which is nothing like human tactics or strategy. It remains desirable to try to understand and mimic the way humans learn and play. A big open question is whether the future of game playing software, or software in general, will adopt these strategies. The human brain trades off search speed and accuracy for robustness and possibly scalability. Human decisions may sometimes be inferior, but they rarely exhibit the catastrophic failures resulting from software bugs. The extent to which the strategies of human expertise and computer algorithms hybridise will be one of the really exciting topics of the next decade.

ACKNOWLEDGEMENT

This work was supported by the Australian Research Council under Discovery Project DP0881829 and the US Airforce under grant 104116.

REFERENCES

- [1] F. Gobet, *Moves in Mind; The Psychology of Board Games*. Psychology Press, Sep. 2004. [Online]. Available: <http://www.worldcat.org/isbn/1841693367>
- [2] A. D. Groot and F. Gobet, *Perception and memory in chess: Heuristics of the professional eye*. Assen: Van Gorcum, 1996.
- [3] F. Gobet, P. Lane, S. Croker, P. Cheng, G. Jones, I. Oliver, and J. Pine, "Chunking mechanisms in human learning," *Trends in Cognitive Sciences*, vol. 5, pp. 236–243, 2001.
- [4] K. Ericsson and N. Charness, "Expert performance: its structure and acquisition," *American Psychologist*, vol. 49, pp. 725–7247, 1994.
- [5] M. Harré, T. Bossomaier, and A. Snyder, "The development of human expertise in a complex environment," *Minds Mach.*, vol. 21, pp. 449–464, August 2011. [Online]. Available: <http://dx.doi.org/10.1007/s11023-011-9247-x>
- [6] X. Cai and D. Wunsch, "Computer Go: A grand challenge to AI," in *Challenges for Computational Intelligence*. Springer Berlin, 2007, pp. 443–465.
- [7] M. Campbell, A. Hoane, and F. Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.
- [8] Harré, M. S., Bossomaier, T., Gillett, A., and Snyder, A., "The aggregate complexity of decisions in the game of go," *Eur. Phys. J. B*, vol. 80, no. 4, pp. 555–563, 2011. [Online]. Available: <http://dx.doi.org/10.1140/epjb/e2011-10905-8>
- [9] R. Wicks, S. Chapman, and R. Dendy, "Mutual information as a tool for identifying phase transitions in dynamical complex systems with limited data," *Phys. Rev. E*, vol. 75, 2007.
- [10] C. Shannon and W. Weaver, *The Mathematical Theory of Communication*. Univ. Ill. Press, Urbana, 1949.
- [11] S.-J. Gu, C.-P. Sun, and H.-Q. Lin, "Universal role of correlation entropy in critical phenomena," *Journal of physics A*, 5 2006.
- [12] M. Harré and A. Snyder, "Intuitive expertise and perceptual templates," *Minds and Machines*, pp. 1–16, 2011, 10.1007/s11023-011-9264-9. [Online]. Available: <http://dx.doi.org/10.1007/s11023-011-9264-9>
- [13] F. Gobet and P. Chassy, "Expertise and intuition: a tale of three theories," *Minds and Machines*, vol. 19, pp. 151–180, 2009.
- [14] W. Chase and H. Simon, "The mind's eye in chess," in *Visual Information Processing*, C. W.G., Ed. Academic Press, NY, 1973, pp. 215–281.
- [15] F. Gobet and H. Simon, "Five seconds or sixty? presentation time in expert memory," *Cognitive Science*, vol. 24, pp. 651–682, 2000.
- [16] J. Rubin and I. Watson, "A memory-based approach to two-player texas hold'em," in *AI 2009: Advances in Artificial Intelligence, Proceedings*, ser. Lecture Notes in Artificial Intelligence, A. Nicholson and X. Li, Eds. Springer, 2009, vol. 5866, pp. 465–474, 22nd Australian Joint Conference on Artificial Intelligence DEC 01-04, 2009 Melbourne, Australia.
- [17] J. Tromp and G. Farneback, "Combinatorics of Go," *Computers and Games*, pp. 84–99, 2007.
- [18] S. Gelly and Y. Wang, "Exploration exploitation in Go: UCT for Monte-Carlo Go," in *Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006)*. Citeseer, 2006.
- [19] D. Stern, R. Herbrich, and T. Graepel, "Bayesian pattern ranking for move prediction in the game of go," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 873–880.
- [20] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, pp. 59–69, 1982, 10.1007/BF00337288. [Online]. Available: <http://dx.doi.org/10.1007/BF00337288>
- [21] D. Kahneman, *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

An Adaptive Computational Intelligence Algorithm for Simulation-driven Optimization Problems

Yoel Tenne
Formerly with the Department
of Mechanical Engineering and
Science,
Faculty of Engineering,
Kyoto University,
Kyoto, Japan
email: ytennex-e04@yahoo.com

Kazuhiro Izui
Department of Mechanical
Engineering and Science,
Faculty of Engineering,
Kyoto University,
Kyoto, Japan
email: izui@prec.kyoto-u.ac.jp

Shinji Nishiwaki
Department of Mechanical
Engineering and Science,
Faculty of Engineering,
Kyoto University,
Kyoto, Japan
email: shinji@prec.kyoto-u.ac.jp

Abstract—Modern engineering design optimization often evaluates candidate designs with computer simulations. In this setup, there will often exist candidate designs which cause the simulation to fail and would have no objective value assigned to them. This, in turn, can degrade the effectiveness of the design optimization process and lead to a poor final result. To address this issue, this paper proposes a new computational intelligence optimization algorithm which incorporates a classifier into the optimization process. The latter predicts which candidate designs are expected to cause a simulation failure, and its prediction is used to bias the search towards candidate designs for which the simulation is expected to succeed. However, the effectiveness of this approach depends on the classifier being used, but it is typically not known a-priori which classifier best suits the problem being solved. To address this issue, the proposed algorithm employs a statistically rigorous procedure to autonomously select the classifier type, and to adjust the classifier selection procedure with the goal of improving its accuracy. A performance analysis with a simulation-driven design problem demonstrates the effectiveness of the proposed algorithm.

Index Terms—*expensive optimization problems; computational intelligence; modelling; classification; model selection.*

I. INTRODUCTION

Nowadays engineers often use *computer simulations* to evaluate candidate designs, with the goal of reducing the duration and cost of the product design process. Such simulations, which must be properly validated with laboratory experiments, transform the design process into an optimization problem having three distinct features [2]:

- The simulation acts as the objective function, namely, it assigns candidate designs their corresponding objective values. However, the simulation is often a legacy code or a commercial software whose inner workings are inaccessible to the user, and so an analytic expression for this function is unavailable. Such a *black-box function* precludes the use of optimizers which require an analytic function.
- Each simulation run is *computationally expensive*, that is, it requires considerable computer resources, and this severely restricts the number of candidate designs which can be evaluated.

- Both the real-world physics being modelled, and the numerical simulation process, may result in an objective function having a complicated nonconvex landscape, which makes it difficult to locate an optimum.

Accordingly, such optimization scenarios are commonly termed in literature as *expensive black-box optimization problems* [2].

A framework which has proven effective in such challenging problems is that of *metamodel-assisted computational intelligence (CI) algorithms*. It combines a *metamodel* which approximates the expensive black-box function and provides predicted objective values at a much lower computational cost, with a *CI optimizer* which seeks an optimum of the metamodel. Due to its explorative nature, a *CI optimizer* often performs well in challenging nonconvex landscapes.

While the above optimization framework has proven effective, simulation-driven optimization problems often present another challenge, namely, some candidate designs will cause the simulation to fail, and would therefore not provide the expected objective value. We refer to such designs as *simulator-infeasible (SI)*, while those for which the simulation completes successfully are termed *simulator-feasible (SF)*. *SI designs* have two main implications on the optimization search:

- Since they do not have a corresponding objective value, the objective function becomes discontinuous, and this exacerbates the difficulty of the optimization search.

and

- Such designs can consume a large portion of the allotted computational resources without providing any objective values, and can therefore degrade the search effectiveness and lead to a poor final result.

A fundamental assumption in this study is that the simulation failures are caused by an unknown limitation of the simulation code, and that they are not random. This implies that repeated evaluations of a *SF candidate solution* will consistently succeed, while repeated evaluations of a *SI candidate solution* will consistently fail. Limitations of the simulation code can be attributed to a variety of reasons, for example, the inability

to handle complex geometries, or an attempt to simulate physical conditions which are not supported by the numerical approximations employed in the simulation.

Based on the description so far, we summarize the underlying settings and core assumptions on which this paper is based:

- The optimization problem involves a black-box objective function which is computationally expensive to evaluate.
- The black-box objective function may have a complicated nonconvex landscape which exacerbates the optimization difficulty.
- Some candidate solutions will cause the simulation to fail, namely it will return no objective value. Such failures are nonrandom, but their cause is unknown.

Numerous studies have referred to such simulation failures and the difficulties they introduce into the optimization search, for example, Büche et al. [3], Okabe [4], and Poloni et al. [5]. The multitude of such references indicates that SI candidate designs are common in real-world applications, and therefore that it is important to effectively handle them. Two main strategies for handling SI vectors include discarding such vectors altogether, or assigning them a penalized objective value and then incorporating them into the metamodel. However, both of these strategies have significant demerits, for example, they discard information which can be beneficial to the search, or they result in a metamodel whose landscape is severely deformed.

In these settings, this study proposes a new approach in which a metamodel-assisted CI algorithm incorporates a *classifier* into the optimization search. The role of the classifier is to predict if a candidate design is SI or not, and its prediction is then used to bias the search towards candidate designs predicted to be SF. However, the effectiveness of this approach depends on the type of classifier being used. Typically, it is not known prior to the optimization search which classifier best suits the problem being solved, while an unsuitable classifier can degrade the search effectiveness. To circumvent this, this study employs a procedure which autonomously selects the most suitable classifier type during the search, based on the statistical procedure of *cross-validation* (CV). To further enhance this procedure, the proposed algorithm also calibrates during the search the *split ratio* parameter related to this procedure.

To the best of our knowledge, such a computational intelligence algorithm which incorporates a metamodel and a classifier, and which autonomously selects the classifier type and calibrates the CV procedure, is new. To evaluate its effectiveness, the proposed algorithm was tested using a representative simulation-driven problem of airfoil shape optimization. Analysis of the test results demonstrates the effectiveness of the proposed algorithm, and the contribution of the proposed classifier selection procedure.

The remainder of this paper is as follows: Section II provides the pertinent background information, Section III describes in detail the proposed algorithm, and Section IV

provides an extensive performance analysis. Lastly, Section V concludes this paper.

II. BACKGROUND

This section provides background information on expensive optimization problems, SI vectors in optimization, and statistical accuracy estimation.

A. Expensive optimization problems

As mentioned in Section I, expensive optimization problems are common in engineering, and Figure 1 shows the layout of such problems in which the simulation is viewed as a black-box function, namely, it assigns objective values to candidate designs, while its analytic expression is unknown. In this setup, the candidate designs are represented as vectors of design variables, and are provided as inputs to the simulation. Overall, such optimization problems arise in domains ranging from the design of electronic devices to the design of aircraft, and a representative problem is described in Section IV-A.

Also as mentioned, the resultant objective function often has a complicated, nonconvex landscape, which can lead gradient-based optimizers to converge to a poor final result. This has motivated the use of CI optimizers in such problems, as they tend to be more explorative, and hence often perform better in complicated nonconvex objective landscapes. Such optimizers typically employ a *population* of candidate solutions and manipulate them using a variety of operators. One such widely used CI optimizer, which is also employed in this study, is the *evolutionary algorithm* (EA), whose mechanics are inspired by the paradigms of adaptation and survival of the fittest. A baseline EA applies the following operators [6]:

- Selection: The candidate solutions (vectors) with the best objective value are selected as *parents*.
- Recombination: Two parents are selected, and their vectors are combined to yield an offspring. This is repeated several times to generate a population of offspring.
- Mutation: Offspring are selected at random, and some of their vector components are randomly changed.

The offspring population is then evaluated, and the fittest candidate solutions, namely, those with the best objective values, are taken to be the population of the next ‘generation’. The process then repeats until a termination criterion is met, for example, if the maximum number of generations has been reached. Through these operators, the EA drives the population to adapt to the function landscape, and to converge to an optimum. While the above description is representative of many EAs in literature, other variants have been proposed which may employ different operators. Algorithm 1 gives a pseudocode of a baseline EA.

Since CI optimizers directly evaluate candidate solutions and do not use gradient information, they often require many thousands of function evaluations to yield a satisfactory solution. This is a major obstacle in applying them to expensive optimization problems, where the objective function can be evaluated only a small number of times. As mentioned in Section I, an established framework to circumvent this is to

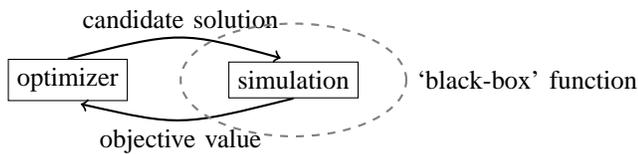


Fig. 1. The layout of an expensive black-box optimization problem. The optimizer generates candidate solutions, and these are evaluated by the simulation to obtain their corresponding objective values. The optimizer views the simulation as a black-box function, that is, having no analytic expression.

Algorithm 1: A baseline evolutionary algorithm (EA)

```

initialize a population of candidate solutions;
evaluate each candidate solution in the population;
/* main loop */
repeat
  select a group of candidate solutions and designate
  them as parents;
  recombine the parents to create offspring;
  mutate some of the offspring;
  evaluate the offspring;
  select the candidate solutions which will comprise
  the population of the next generation;
until convergence, or maximum number of generations
reached;
```

employ a metamodel which approximates the true expensive function and provides the optimizer with predicted objective values at a much lower computational cost. Metamodels are typically interpolants trained with previously evaluated vectors, and variants include artificial neural networks, Kriging, polynomials, and radial basis functions (RBF) [7]. Numerous metamodel-assisted CI algorithms have proposed, and examples include the Kriging assisted EA by Ratle [8] which is also described in Section IV-C, and Booker et al. [9] which coupled a pattern search optimizer with quadratic metamodels. Later examples include Emmerich et al. [10] which used an evolutionary strategies (ES) optimizer coupled with a Kriging metamodel, and Liang et al. [11] which coupled an EA with a least-square fit polynomial metamodel. Poloni et al. [5] and Muyl et al. [12] studied algorithms coupled with an artificial neural networks (ANN). Later, Büche et al. [3] studied an ES optimizer assisted with a Kriging metamodel, and employed an elaborate sampling scheme which sought solutions based on different trade-offs of exploration-exploitation, as described in Section IV-C. More recent examples include Tenne and Armfield [13], Neri et al. [14] and Zhou et al. [15]. Given the established effectiveness of the metamodeling framework, it is also employed in this study.

While metamodels address the issue of computationally expensive evaluations, they introduce the challenge of *prediction inaccuracy*. Specifically, due to the restricted number of expensive function evaluations, only a small number of vectors will be available to train the metamodel, which degrades its accuracy. In severe cases, the optimizer may even converge

to a *false optimum*, namely, an optimum of the metamodel which is not an optimum of the true expensive function [16], and it is therefore necessary to safeguard the metamodel accuracy to ensure the progress of the optimization search. The proposed algorithm accomplishes this by leveraging on the *trust-region* (TR) approach which originated in the field of nonlinear programming [17], where initially a *trial step* is performed to seek an optimum of the metamodel in the TR, namely, the region where the metamodel is assumed to be accurate. Next, the TR and metamodel are updated based on the optimum found, and the process repeats until a termination condition is met. A merit of the TR approach is that it ensures asymptotic convergence to an optimum of the true expensive function [17]. Section III gives a detailed description of the TR approach implemented in this study.

B. Simulator-infeasible vectors

As mentioned in Section I, this study focuses on expensive optimization problems with simulator-infeasible (SI) vectors, namely, which cause the simulation to fail. A multitude of studies have referred to such vectors and to the difficulties they introduce into the optimization search. For example, Poloni et al. [5] described an optimization problem which involved a computational fluid dynamics analysis, and noted that some candidate designs caused “failure of the simulation code”. In another study, Booker et al. [9] described a rotor blade structural optimization problem in which “attempts to evaluate the objective function failed”. Similarly, Büche et al. [3] described an aerodynamics shape optimization problem in which “evaluation of all points fails”. Additional pertinent studies include Liang et al. [11], Conn et al. [18] and Okabe [4].

Several techniques have been explored in an effort to handle SI vectors. For example, Rasheed et al. [19] described an aircraft design optimization problem in which an EA directly called the expensive simulation, and no metamodels were employed. A classifier was used to screen candidate design prior to the simulation call, and those predicted to be SI were assigned a ‘death penalty’, namely, a fictitious and highly penalized objective value, to quickly eliminate them from the population, but no metamodels were employed. In another related study, Emmerich et al. [10] also used the penalty approach, but incorporated the penalized vectors into the metamodel in an attempt to bias the search towards SF vectors. In contrast, Büche et al. [3] discarded the SI vectors altogether, so that the metamodel was trained using only the SF vectors.

These strategies, and similar ones, have several demerits in the context of expensive optimization problems: a) assigning SI vectors a penalized objective value and then incorporating them into the metamodel can severely deform the metamodel landscape and degrade its accuracy, while b) discarding SI vectors results in a loss of information which might have been useful in enhancing the optimization search. As an example, Figure 2 shows the effect of penalizing SI vectors and incorporating them into a Kriging metamodel, which is

described in Section III. Figure 2(a) shows the metamodel resulting from a sample of 30 SF vectors, while Figure 2(b) shows the resultant metamodel when 20 SI vectors were added to the baseline sample and were assigned the worst objective value from the baseline sample. The metamodel landscape was severely deformed and consequently locating an optimum of the true objective function became more difficult.

Such issues have motivated exploring alternative approaches for handling SI vectors. For example, Tenne and Armfield [20] proposed an approach which employed two metamodels, where one was used for approximating the objective function and another for generating a penalty which was based on the distance of a new candidate solution to previously encountered SI ones. Other studies have examined using classifiers for constrained non-linear programming, though unrelated to SI vectors [21]. Further exploring the use of classifiers, Tenne et al. [22] obtained preliminary results with a classifier-assisted algorithm for handling SI vectors. However, the algorithm used a single type of classifier, and it did not attempt to select the classifier during the search. Recently, Tenne et al. [1] presented a preliminary investigation on a framework which adapts the classifier type based on the problem being solved. The present study leverages on the latter framework and extends it by proposing to also adapt the CV split ratio used in the classifier selection step. The present study also provides a more extensive performance analysis.

C. Accuracy estimation

As mentioned in Section I, the proposed algorithm employs a classifier to predict which candidate designs will cause the simulation to fail, and to improve the effectiveness of this approach, it selects the classifier deemed most accurate out of a family of candidates.

Accuracy estimation is rigorously addressed in the general statistical framework of *model selection*, in which a *model* refers to any functional relation which is used to explain an inputs–outputs relation [23]. In the model selection procedure, several candidate models are prescribed and their accuracy is estimated, after which the model deemed as the most accurate is selected as the optimal one. An established procedure for estimating the model accuracy is that of *cross-validation* (CV), in which the sample of vectors is split into a *training sample* and a *testing sample*. A candidate model is trained using the former, and its predictions for the testing vectors are compared to their already known exact function values.

The CV procedure relies on the *split ratio* parameter, which determines which portion of data set will be designated as the training sample and which as the testing sample. This suggests that the accuracy of the procedure will be affected by the split ratio used. To verify this, the CV procedure was used to estimate the accuracy of two candidate classifiers, namely, *k nearest neighbours* (*kNN*) and *support vector machine* (*SVM*), whose details are given in Appendix A, and the tests were performed using the two well-established data sets *iris* and *yeast* provided by Frank and Asuncion [24]. The accuracy measure used was the total classification error, namely, the

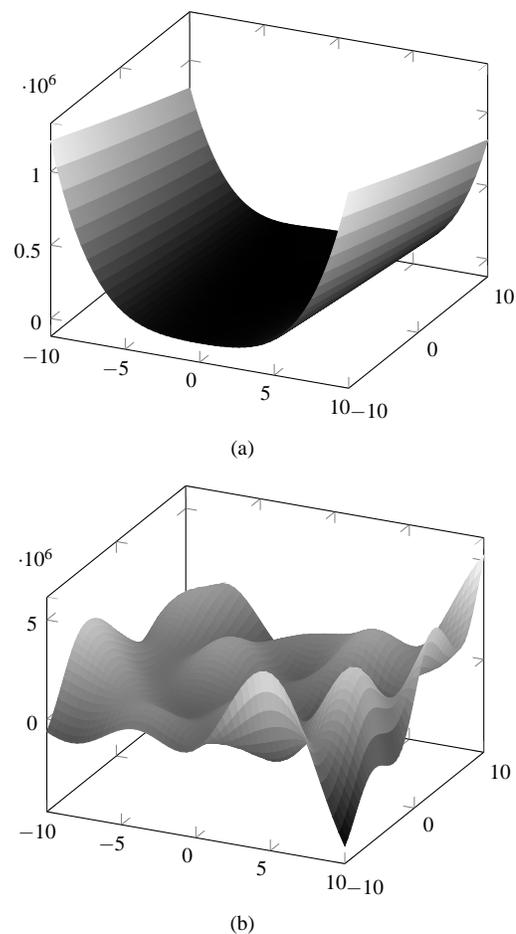


Fig. 2. An example of the effect of SI vectors on the metamodel. The objective function was Rosenbrock, whose optimum is at (1, 1). (a) shows a Kriging metamodel trained using a sample of 30 SF vectors, and (b) shows the resultant metamodel when 20 SI vectors were added to the sample and were assigned the worst objective value of the sample in (a). The landscape of the resultant metamodel was severely deformed, and the optimum of the true objective function was masked.

number of items in the testing sample to which the classifier assigned an incorrect class. To check the effect of different split ratios, the full data set was initially split in a 80–20 training–testing ratio, and the accuracy of each classifier was estimated. The accuracy estimates from this step are considered as the reference results, since they employed the full data set. Next, the training sample was used as the baseline sample, and the accuracy of each classifier was estimated by using each of the following training–testing split ratios in turn: 0.8–0.2, 0.5–0.5, and 0.2–0.8. Table I shows the test results and the rankings of the two classifiers. It follows that the rankings corresponding to the 0.5–0.5 and 0.2–0.8 split ratios matched those obtained with the full sample, while those of the 0.8–0.2 split ratio differed. This in turn verifies the above assumption, namely, that the split ratio affected the accuracy of the CV procedure.

Since the optimal split ratio is unknown prior to the accuracy estimation step, it is possible that an unsuitable value would be used, which in turn would degrade the accuracy of

the CV procedure. To circumvent this, the proposed algorithm employs a procedure to autonomously select a suitable split ratio, as described in Section III.

III. PROPOSED ALGORITHM

This section describes the proposed algorithm in detail, and explains how it addresses the optimization challenges discussed in Sections I and II. The algorithm leverages on three paradigms:

- Classification of candidate vectors: Each candidate vector is treated as having two attributes, namely, its *objective value*, which is predicted by a metamodel, and its *class*, namely, if it is SI or SF, which is predicted by a classifier.
- Selection of the classifier type: Typically, it is not known prior to the optimization search which classifier type is most suitable to the problem being solved. To circumvent this, during the optimization search the proposed algorithm uses the CV procedure to autonomously select the most suitable type of classifier. To further improve the accuracy of this approach, the proposed algorithm also continuously selects during the search the most suitable split ratio value.
- Trust-region (TR) optimization: Given the inherent metamodel inaccuracy, a TR framework is employed to ensure convergence to an optimum of the true expensive function.

The proposed algorithm operates in five main steps: initialization, training a metamodel, selecting the classifier type and training a corresponding classifier, performing a TR trial step to seek an optimum, and performing the TR updates. The details of these steps are as follows:

Step 1) *Initialization*: The proposed algorithm begins by generating an initial sample of vectors using a Latin hypercube (LH) design of experiments [25]. This is a statistically oriented sampling method which ensures that the sample is space-filling, namely, that the vectors are distributed throughout the search space, which improves the accuracy of the resultant metamodel. A sample of s vectors is generated as follows. The range of each variable is split into s equally sized intervals, and one point is sampled at random in each interval. Next, a sample point is selected at random and without replacement for each variable, and these samples are combined to produce a vector. This sequence is repeated for s times to create the

TABLE I
CLASSIFIER ACCURACY RANKINGS BY DIFFERENT CV SPLIT RATIOS

Split ratio	Error	Rank	Error	Rank
Full	2	1	3	2
0.8	1	2	0	1
0.5	1	1	2	2
0.2	5	1	7	2

Highlighted lines have the same ranks as those obtained with the full sample.

complete sample, which is then evaluated with the expensive simulation and is stored in memory. After this step, the main optimization loop begins.

Step 2) *Metamodel training*: In this step, the proposed algorithm trains a metamodel by using the SF vectors stored in memory and ignores the SI vectors. In this study a Kriging metamodel was employed, based on its prevalence in literature [3, 26, 27]. This metamodel is statistically-oriented and combines two components: a ‘drift’ function, which is a global coarse approximation of the true expensive function, and a local correction based on the correlation between the interpolation vectors. Given a set of evaluated vectors, $\mathbf{x}_i \in \mathbb{R}^d$, $i = 1 \dots n$, the Kriging metamodel is trained such that it exactly interpolates the observed values, that is, $m(\mathbf{x}_i) = f(\mathbf{x}_i)$, where $m(\mathbf{x})$ and $f(\mathbf{x})$ are the metamodel and true objective function, respectively. Using a constant drift function [28] gives the Kriging metamodel

$$m(\mathbf{x}) = \beta + \kappa(\mathbf{x}), \quad (1)$$

with the drift function β and local correction $\kappa(\mathbf{x})$. The latter is defined by a stationary Gaussian process with mean zero and covariance

$$\text{Cov}[\kappa(\mathbf{x})\kappa(\mathbf{y})] = \sigma^2 c(\theta, \mathbf{x}, \mathbf{y}), \quad (2)$$

where $c(\theta, \mathbf{x}, \mathbf{y})$ is a user-prescribed correlation function. A common choice for the latter is the Gaussian correlation function [28], defined as

$$c(\theta, \mathbf{x}, \mathbf{y}) = \prod_{i=1}^d \exp(-\theta(x_i - y_i)^2), \quad (3)$$

and combining it with the constant drift function transforms the metamodel from (1) into the following form

$$m(\mathbf{x}) = \hat{\beta} + \mathbf{r}(\mathbf{x})^T \mathbf{R}^{-1}(\mathbf{f} - \mathbf{1}\hat{\beta}). \quad (4)$$

Here, $\hat{\beta}$ is the estimated drift coefficient, \mathbf{R} is the symmetric matrix of correlations between all interpolation vectors, \mathbf{f} is the vector of objective values, and $\mathbf{1}$ is a vector with all elements equal to 1. \mathbf{r}^T is the correlation vector between a new vector \mathbf{x} and the sample vectors, namely,

$$\mathbf{r}^T = [c(\theta, \mathbf{x}, \mathbf{x}_1), \dots, c(\theta, \mathbf{x}, \mathbf{x}_n)]. \quad (5)$$

The estimated drift coefficient $\hat{\beta}$ and variance $\hat{\sigma}^2$ are obtained from

$$\hat{\beta} = (\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1})^{-1} \mathbf{1}^T \mathbf{R}^{-1} \mathbf{f}, \quad (6a)$$

$$\hat{\sigma}^2 = \frac{1}{n} \left[(\mathbf{f} - \mathbf{1}\hat{\beta})^T \mathbf{R}^{-1} (\mathbf{f} - \mathbf{1}\hat{\beta}) \right]. \quad (6b)$$

Fully defining the metamodel requires the correlation parameter θ , whose optimal value, θ^* , is typically taken as the maximizer of the metamodel likelihood. In practise, the latter is obtained by minimizing the negative log-likelihood, namely

$$\theta^* : \min - (n \log(\hat{\sigma}^2) + \log(|\mathbf{R}|)). \quad (7)$$

While a different correlation parameter can be used for each variable, this study follows the practise prevalent in literature in which the metamodel employs a single correlation parameter. This results in a univariate likelihood function, which is relatively easy to optimize.

Step 3) *Classifier selection and training*: In the next step, the proposed algorithm trains a classifier to predict if a vector is SF or SI. To further improve this technique, the proposed algorithm employs the CV to select during the search a classifier deemed as most suitable out of a family of candidates. To further enhance the accuracy of this procedure, the proposed algorithm employs an additional step to identify a suitable split ratio for the CV procedure, out of a prescribed set of candidate ratios. The details of the procedure are as follows:

- 3.1) The set of vectors stored in memory is split into sample A and sample B in a 80–20 split ratio.
- 3.2) Using only sample A, the proposed algorithm loops over the prescribed set of candidate split ratios, s_i , $i = 1 \dots n_s$, where n_s is the number of candidate of ratios, and for each it performs the following steps:
 - 3.2.1) It generates a training sample and a testing sample based on sample A.
 - 3.2.2) For each candidate type of classifier, the proposed algorithm trains a corresponding classifier using the training sample, and then estimates the classifier's accuracy by using the testing sample, where the accuracy measure is the *total classification error*, defined as

$$e = \sum_{i=1}^l (\hat{c}(\mathbf{x}_i) \neq F(\mathbf{x}_i)), \quad (8)$$

where \mathbf{x}_i , $i = 1 \dots l$, are the vectors in the testing sample, $\hat{c}(\mathbf{x})$ is the prediction of the classifier which was trained using the training sample, and $F(\mathbf{x}_i)$ is the true and known class of the testing vectors. For the latter, $F(\mathbf{x}_i) = 1$ was used for a SF vector, and $F(\mathbf{x}_i) = -1$ for a SI vector.

- 3.2.3) The candidate classifiers are ranked based on their obtained total classification errors, which yields a vector of ranks \mathbf{r}_i , where i is the index of the current split ratio being considered.
- 3.3) After completing the above procedure for all candidate split ratios, the proposed algorithm loops over the set of candidate classifier types, and using the samples A and B obtained in step 3.1, it trains a classifier using sample A, and estimates the classifier's accuracy using sample B. The classifier types are ranked based on their

estimated accuracies, which yields a vector of ranks \mathbf{r}_0 .

- 3.4) The proposed algorithm selects as the optimal split ratio (s^*) the one whose corresponding ranks vector \mathbf{r}_i is most similar to the reference ranks vector \mathbf{r}_0 . This similarity is measured by the l_1 norm, namely

$$s^* = s_{i^*}, \quad i^* : \min_{i=1 \dots n_s} \|\mathbf{r}_i - \mathbf{r}_0\|_1, \quad (9)$$

where i^* is the index of the optimal split ratio. The basis for this procedure is that the most suitable split ratio should yield a ranks prediction which is relatively insensitive to the sample size. Therefore, the ranks vectors obtained in step 3.2.3, namely, \mathbf{r}_i , $i = 1 \dots n_s$, which were obtained based on the training sample derived from sample A, are compared to the ranks vector from step 3.4, namely, \mathbf{r}_0 , which was obtained based on the full set of vectors stored in memory.

- 3.5) After identifying the most suitable split ratio, the proposed algorithm selects the classifier type which had the lowest prediction error in the selected split ratio, and trains a classifier, designated as $c(\mathbf{x})$, using all the vectors stored in memory. This classifier is then used in the optimization search performed in step 4.

In this study the proposed algorithm selected between three classifiers types: *k nearest neighbours* (kNN), *linear discriminant analysis* (LDA), and *support vector machine* (SVM), whose details are given in Appendix A. The candidate values for the training-testing split ratios were 0.8–0.2, 0.5–0.5, and 0.2–0.8.

- Step 4) *TR trial step*: The best vector in the memory storage is taken as the TR centre (\mathbf{x}_b), and a TR trial-step is performed, namely, an optimizer is invoked to find an optimum in the bounded region

$$\mathcal{T} = \{\mathbf{x} : \|\mathbf{x} - \mathbf{x}_b\|_2 \leq \Delta\}, \quad (10)$$

where Δ is the TR radius. The optimizer used is the real-coded EA of Chipperfield et al. [29], which follows the setup described in Section II-A, namely, it begins by selecting a set of parents, recombines them to produce offspring, mutates some of the offspring, and selects the population of the next generation from the union of the offspring and the best parents. Table II gives the complete parameter settings of this EA, which are based on those suggested in literature [29, 30].

During the trial step, the EA uses the following *modified objective function* which combines the prediction of the metamodel from Step 2 and the classifier from Step 3, as follows

$$\hat{m}(\mathbf{x}) = \begin{cases} m(\mathbf{x}) & \text{if } c(\mathbf{x}) \text{ is SF} \\ p & \text{if } c(\mathbf{x}) \text{ is SI} \end{cases} \quad (11)$$

TABLE II
INTERNAL PARAMETERS OF THE EA UTILIZED IN THIS STUDY [29]

Population size	100
Generations	100
Selection	Stochastic universal selection (SUS)
Recombination	Intermediate, applied with probability $p = 0.7$
Mutation	Breeder Genetic Algorithm (BGA) mutation, applied with probability $p = 0.1$
Elitism	10%

namely, the EA receives the objective value predicted by the metamodel $m(\mathbf{x})$ if the classifier predicts a vector is SF, but it receives the penalized objective value p otherwise. The latter is taken as the worst objective value in the initial LH sample.

This formulation enhances the optimization search in two main aspects. First, the classifier accumulates the information about the SI vectors encountered during the search and uses it to predict the distribution of such vectors in the search space, so this potentially beneficial information is not discarded. Second, the classifier's prediction is used to bias the search but without affecting the metamodel landscape, and this avoids the issues discussed in Section II-B. To visualize the effect of this setup, and to demonstrate the predictions of the metamodel, the classifier, and how they are combined into a single modified objective function, Figure 3 gives a synthetic example with the Rosenbrock function. The plots show that the landscape predicted by the modified objective function closely follows that of the baseline metamodel, and embeds the knowledge on the location of the SI vectors, but it is only minimally deformed.

Step 5) *TR updates*: The optimum found by the EA, \mathbf{x}^* , is evaluated with the true expensive function, which yields the exact objective value $f(\mathbf{x}^*)$. Following the classical TR framework [17], the proposed algorithm performs the following updates:

- If $f(\mathbf{x}^*) < f(\mathbf{x}_b)$: The trial step was successful since the predicted optimum is indeed better than the current best solution, namely, \mathbf{x}_b . Accordingly, the TR is centred at the new optimum, and the TR is enlarged by doubling its radius.
- If $f(\mathbf{x}^*) \geq f(\mathbf{x}_b)$ and there are sufficient SF vectors inside the TR: The search was unsuccessful since the predicted optimum is not better than the current best vector. However, since there are sufficient SF vectors in the TR, the metamodel is deemed as being sufficiently accurate to justify contracting the TR. Accordingly, the TR is contracted by halving its radius.
- If $f(\mathbf{x}^*) \geq f(\mathbf{x}_b)$ and there are insufficient SF vectors inside the TR: The search was unsuccessful, but this may be since the metamodel is inaccurate due to an insufficient number of SF vectors in the TR. Therefore, the algorithm samples new vectors (\mathbf{x}_n) inside the TR, as explained below.

Objective function and sample vectors

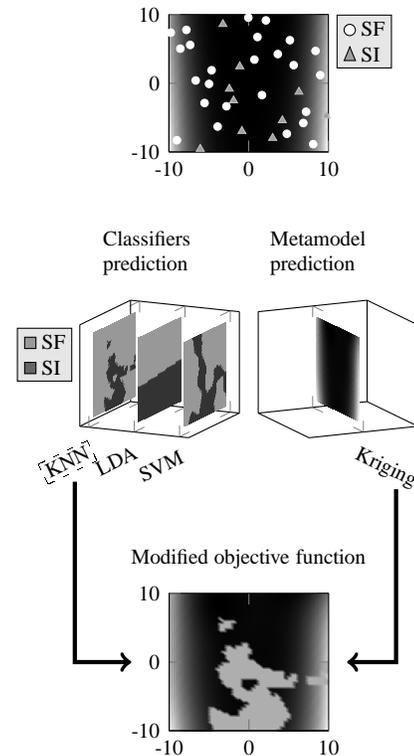


Fig. 3. An example showing how the proposed algorithm generated the modified objective function. The objective function was Rosenbrock, and the sample was comprised of 26 SF vectors and 9 SI vectors. The proposed algorithm trained a Kriging metamodel using the SF vectors, and trained the classifiers using the entire sample. The k NN classifier was deemed as the most accurate, and therefore its prediction was used in the modified objective function. The landscape of the latter was modified based on the predictions regarding SI vectors, but it was only minimally deformed.

As a change from the classical TR framework, the proposed algorithm contracts the TR only if it contains a sufficient number of SF vectors, to avoid a too rapid TR contraction and premature convergence [17]. To select a suitable threshold value (q) for the number of these vectors, numerical experiments have been performed and are described in Section IV-B. Another change from the classical TR framework is the sampling of new vectors to improve the accuracy of the metamodel in the TR. There are two considerations in selecting these vectors: i) they should improve the metamodel accuracy locally around the current optimum, and alternatively ii) they should improve the metamodel accuracy over the entire TR, and particularly in regions sparse with vectors [31]. Since these are typically two opposing considerations, the proposed algorithm generates several new vectors which correspond to different trade-offs between these considerations. The vectors are taken as the minimizers of the following objective function

$$h(\mathbf{x}) = wh_1(\mathbf{x}) + (1-w)h_2(\mathbf{x}), \quad (12)$$

where the minimization is performed by the real-coded EA described earlier. Here, $h_1(\mathbf{x})$ is the rank of the vector \mathbf{x} based on its objective value, such that the best vector in the EA population is assigned a rank of 1, the following one a rank of two, and so on. Also, $h_2(\mathbf{x})$ is the rank of the vector \mathbf{x} based on its distance from existing vectors in the TR, where the vector in the EA population which is farthest is given a rank of one, the vector having the 2nd largest distance is given a rank of two, and so on. The weight w defines the trade-off between the two considerations, where $w = 1$ implies a vector is searched based only on its objective value, which will result in the new vector being in the vicinity of the TR centre, while $w = 0$ implies a vector is searched based only on its distance to existing vectors in the TR, which will result in a vector being away from existing vectors. Equation (12) uses a rank based approach to make the search more consistent across different objective functions. To identify suitable weights, numerical experiments have been performed and are described in Section IV-B.

To complete the algorithm description, several additional points are noted:

- While in the description above the proposed algorithm used a Kriging metamodel and selected between a k NN, linear discriminant analysis (LDA), and SVM classifiers, other types of metamodels and classifiers can be readily used.
- To avoid a numerical breakdown of the metamodel training process, the proposed algorithm evaluates a new vector and adds it to the memory storage only if it differs from those already stored.
- There is some computational overhead introduced by the proposed algorithm due to the classifier selection step. However, since no expensive evaluations are involved in this step, and since the classifiers' training phase is computationally cheap, the overhead is minimal.

To complete the description of the proposed algorithm, Figure 4 gives a schematic layout of its optimization iteration, and Algorithm 2 gives its pseudocode.

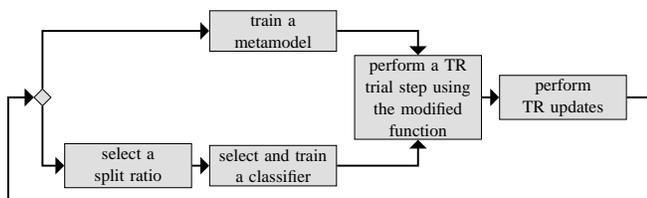


Fig. 4. The layout of an optimization iteration of the proposed algorithm. The iteration begins by training a metamodel and classifier, including selection of the classifier type and CV split ratio. This is followed by a TR trial step to locate an optimum of the modified objective function, and lastly updates of the TR to ensure the progress of the search.

Algorithm 2: Proposed optimization algorithm

```

/* initialization */
generate an initial LH sample;
evaluate the sample vectors and store in memory;
/* main optimization loop */
repeat
  /* train a metamodel */
  use the SF vectors stored in memory to train a metamodel;
  /* select and train a classifier */
  split the vectors stored in memory into a sample A and a sample B;
  for each candidate split ratio do
    split sample A into a training sample and testing sample using the candidate split ratio;
    for each candidate classifier type do
      train a classifier using the training sample;
      estimate the classifier accuracy using the testing sample;
    rank the classifiers based on their accuracies;
  for each candidate classifier type do
    train a classifier using sample A;
    estimate the classifier accuracy using sample B;
  rank the classifiers based on their accuracies to obtain a ranks vector  $\mathbf{r}_0$ ;
  select the split ratio  $s^*$  whose corresponding ranks vector is most similar to  $\mathbf{r}_0$ ;
  select the classifier type which produced the lowest prediction error when  $s^*$  was used, and train a classifier using all vectors stored in memory;
  /* perform a TR trial step */
  set the TR centre to the best vector stored in memory;
  use a real-coded EA to find an optimum of the modified objective function in the TR;
  /* perform TR updates */
  evaluate the predicted optimum with the true expensive function;
  if the new optimum is better than the best vector in memory then
    double the TR radius
  else if the new optimum is not better than the best vector in memory and there are sufficient vectors in the TR then
    halve the TR radius;
  else if the new optimum is not better than the best vector in memory and there are insufficient vectors in the TR then
    add new vectors in the TR to improve the metamodel accuracy;
    add to the memory storage all the new vectors evaluated with the true expensive function;
until maximum number of analyses completed;
  
```

IV. NUMERICAL EXPERIMENTS

This section describes the numerical experiments used to evaluate the performance of the proposed algorithm. It begins by describing the test problem employed, it then describes a parameter sensitivity analysis which was used to select suitable settings for the algorithm parameters, and lastly it describes and analyzes a set of benchmark tests.

A. Problem description

The test problem employed was that of airfoil shape optimization, as it is both simulation-driven and contains SI vectors, as explained below. The formulation of the problem is as follows. During flight, an aircraft generates *lift*, namely, the force which counters the aircraft weight and keeps it airborne, and *drag*, which is an aerodynamic friction force obstructing the aircraft movement. Both the lift and drag result from the flow of air around the aircraft wing whose cross-section is the airfoil. The optimization goal is then to identify an airfoil shape which maximizes the lift and minimizes the drag. In practise, the design requirements for airfoils are specified in terms of the nondimensional lift and drag coefficients, c_l and c_d , respectively, defined as

$$c_l = \frac{L}{\frac{1}{2}\rho V^2 S} \quad (13a)$$

$$c_d = \frac{D}{\frac{1}{2}\rho V^2 S} \quad (13b)$$

where L and D are the lift and drag forces, respectively, ρ is the air density, V is the aircraft speed, and S is a reference area, such as the wing area. Also important is the *angle of attack* (AOA), which is the angle between the aircraft velocity and the airfoil *chord line*, defined as the straight line joining the leading and trailing edges of the airfoil. Figure 5 gives a schematic layout of the airfoil problem.

Candidate airfoils were represented with the Hicks-Henne parameterization [32], in which the profile of a candidate airfoil is defined as

$$y = y_b + \sum_{i=1}^h \alpha_i b_i(x), \quad (14)$$

where y_b is a baseline airfoil profile, taken as the NACA0012 symmetric airfoil, b_i are basis functions, which following [33], are defined as

$$b_i(x) = \left[\sin \left(\pi x \frac{\log(0.5)}{\log(i/(h+1))} \right) \right]^4, \quad (15)$$

and $\alpha_i \in [-0.01, 0.01]$ are coefficients, which are the problem's design variables. Ten basis functions were used for the upper and lower airfoil profile, respectively, resulting in 20 design variables overall. Figure 5 shows the layout of the airfoil problem and the Hicks-Henne parametrization. The lift and drag coefficients of candidate airfoils were obtained by using *XFOIL*, a computational fluid dynamics simulation for analysis airfoils operating in the subsonic regime [34]. Each airfoil evaluation required up to 30 seconds on a desktop computer. To ensure structural integrity, the thickness of an airfoil (t)

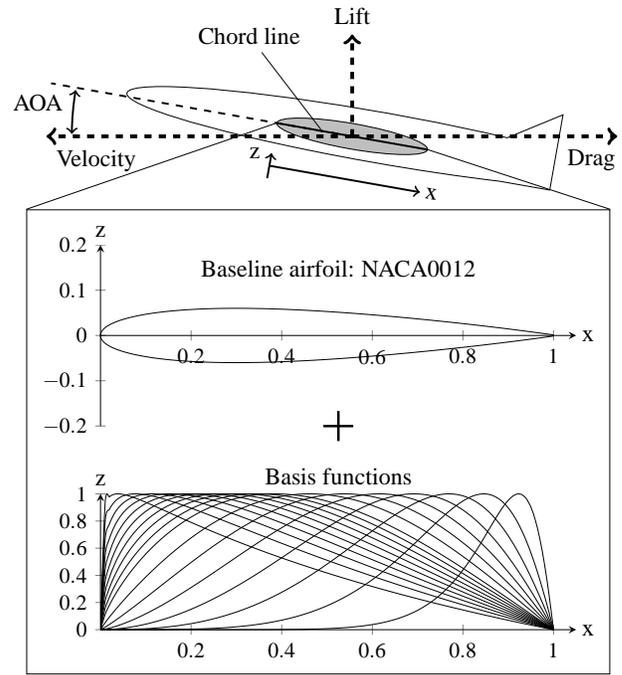


Fig. 5. A schematic layout of the airfoil problem showing the physical quantities involved, and the Hicks-Henne parameterization setup.

between 0.2 to 0.8 of its chord length had to be equal to or larger than a critical value $t^* = 0.1$.

The airfoil shape optimization problem is a pertinent test case since it contains SI vectors, and their prevalence is strongly affected by the angle of attack (AOA) defined earlier. Specifically, since the turbulence of the flow field increases with the AOA, at higher angle values it will be more difficult to complete the aerodynamics analysis, which will result in more simulation failures, and therefore, more SI trial designs. To verify this, 30 different airfoils were sampled and evaluated in identical flight conditions, except for the AOA which was increased from 20° to 50° . Figure 6 shows the obtained results, where, as expected, the number of failed analyses increased with the AOA. Therefore, by changing the AOA we could change the density of SI vectors in the search space, and hence the relative difficulty of the tests. In view of these results, the numerical experiments included three optimization scenarios, namely, with $\text{AOA} = 20^\circ, 30^\circ$, and 40° , which following Figure 6, correspond to a low, medium and high density of SI vectors in the search space, respectively.

B. Parameter sensitivity analysis

As described in Section III, the proposed algorithm relies on two main parameters, namely:

- q : The minimum number of vectors in TR to invoke a TR contraction.

and

- w_i : The weights used for generating new vectors to improve the metamodel accuracy (\mathbf{x}_n).

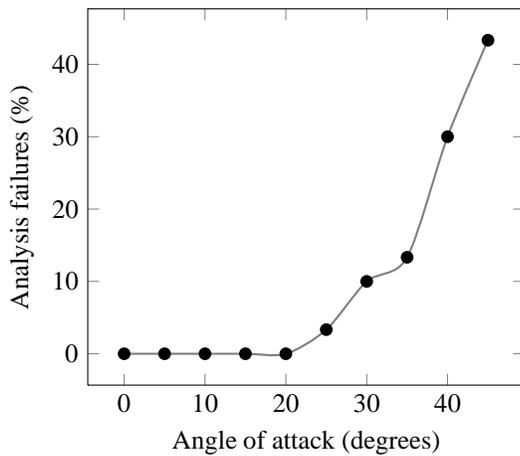


Fig. 6. Simulation failures as a function of the angle of attack (AOA).

To identify suitable values for these parameters, in turn, each parameter was assigned one of three candidate values, and ten optimization runs were repeated with the AOA = 30° setting.

Table III shows the obtained results, where the best mean statistic is emphasized. It follows that suitable parameter settings are a threshold value of $q = 20$, and a set of weights $w = \{0.8, 0.5, 0.2\}$, resulting in three new vectors being added in the TR to improve the metamodel accuracy. Accordingly, these settings were used in the benchmark tests described in the next section.

C. Benchmark tests: Results and analysis

For a comprehensive evaluation, the proposed algorithm was benchmarked against two representative algorithms from literature:

- *EA with Periodic Sampling (EA-PS)* [35]: The algorithm safeguards the metamodel accuracy by periodically evaluating a small subset of the population with

TABLE III

TEST STATISTICS FOR THE PARAMETER SENSITIVITY ANALYSIS

(a) q : Number of vectors in the TR needed to contract the TR

	2 ($=0.1d$)	10 ($=0.5d$)	20 ($=d$)
Mean	9.794e-01	8.772e-01	8.568e-01
SD	5.631e-02	6.540e-02	7.462e-02
Median	9.984e-01	8.674e-01	8.434e-01
Min(best)	8.409e-01	7.875e-01	7.527e-01
Max(worst)	1.028e+00	1.001e+00	1.005e+00

d : Dimension of objective function.

(b) w : Weights used for generating new vectors in the TR

	{0.8,0.2}	{0.8,0.5,0.2}	{0.8,0.6,0.4,0.2}
Mean	9.126e-01	8.741e-01	8.772e-01
SD	6.099e-02	7.245e-02	6.540e-02
Median	9.400e-01	8.847e-01	8.674e-01
Min(best)	7.880e-01	7.175e-01	7.875e-01
Max(worst)	9.661e-01	9.947e-01	1.001e+00

the true objective function, and incorporating them into the metamodel. The algorithm begins by generating an initial sample of vectors (candidate solutions), evaluating them with the expensive function, and training a Kriging metamodel. It then uses a real-coded EA to seek an optimum of the metamodel, where the EA is run for 10 generations. The ten best members of the resultant population are then evaluated with the true expensive function, and are incorporated into the metamodel. This process repeats until the maximum number of expensive function evaluations is reached. In the benchmark tests, the EA was identical to the one used by the proposed algorithm and used the same parameters given in Table II.

- *Expected Improvement with a Covariance Matrix Adaptation Evolutionary Strategies optimizer (EI-CMA-ES)* [3]: The algorithm combines a covariance matrix adaptation evolutionary strategy (CMA-ES) optimizer [36] with a Kriging metamodel, and updates the latter based on the expected improvement framework [37]. The algorithm begins by generating an initial sample of vectors, and evaluates them with the true function. Its main loop then begins, where at each generation it trains a Kriging metamodel by using both the recently evaluated vectors and those stored in memory which are nearest to the best solution. A CMA-ES optimizer is then invoked to locate an optimum of the metamodel in a bounded region defined by the metamodel training sample. In the spirit of the expected improvement framework [37], the function being minimized is

$$\hat{f}(\mathbf{x}) = m(\mathbf{x}) - \rho \zeta(\mathbf{x}), \quad (16)$$

where $m(\mathbf{x})$ is the Kriging metamodel prediction, ρ is a prescribed coefficient, and $\zeta(\mathbf{x})$ is the estimated Kriging prediction error, which is zero at sampled vectors since there the true objective value is known. The search is repeated for $\rho = 0, 1, 2$, and 4, to obtain four solutions corresponding to different search profiles, namely, ranging from a local search ($\rho = 0$) to a more explorative one ($\rho = 4$). All non-duplicate solutions found are evaluated with the true expensive function, and are stored in memory. In case no new solutions were evaluated, for example, because they already match those stored in memory, a new solution is generated by perturbing the current best one. Following Büche et al. [3], the algorithm used a training set of 100 vectors comprising of the 50 most recently evaluated ones and 50 nearest-neighbours, and the CMA-ES used the default settings given in Reference [36].

To also study the contribution of the proposed procedure of selecting the classifier type and calibrating the CV split ratio during the search, the benchmark tests also included the following two variants of the proposed algorithm which were identical to it in operation, except that they used a fixed type of classifier, and therefore did not use the procedures in question: variant i) VK: a variant which used a k NN classifier, and variant ii) VS: a variant which used an SVM classifier.

For all algorithms the limit of simulation calls was set to 200, and their initial sample was generated by a LH procedure and consisted of 20 candidate solutions. To support a valid statistical analysis, 30 runs were repeated for each algorithm at each test case. This was done only for the benchmarking purpose, and is not required in an ordinary optimization search.

For a thorough evaluation, three performance measures were analyzed: i) the final objective value obtained by each algorithm, ii) the number of SI vectors generated by each algorithm during its optimization search, and iii) the classifier type and split ratio which were selected by the proposed algorithm during its optimization search. The details of these analyses are as follows:

- *Final objective value:* To compare the effectiveness of the algorithms, Table IV gives the test statistics of mean, standard deviation (SD), median, best, and worst final objective values obtained by each algorithm in each optimization scenario, with the best mean and median statistics emphasized. The table also gives the significance level (α) at which results of the proposed algorithm were better than those achieved by the other algorithms, where the significance levels considered were 0.01, 0.05, or an empty entry otherwise. Statistical significance was determined using the Mann–Whitney nonparametric test [38, p.423–432].

It follows that the proposed algorithm consistently performed well, as indicated by its mean and median statistics. Also, statistical significance comparisons show that it outperformed the two reference algorithms from literature, namely, EA–PS and EI–CMA–ES in the AOA=20° case, and outperformed the EA–PS algorithm in the AOA=30° and 40° cases. In contrast, the EA–PS algorithm typically achieved either the worst or second worst mean statistic, which highlights the demerit of the penalty approach it employs, namely, that incorporating penalized vectors into the training sample can result in deformation of the metamodel landscape and consequently degrade the search effectiveness. It is noted that the performance gains achieved by the proposed algorithm varied depending on the problem setting (the AOA), and were more modest in the high AOA settings where the high prevalence of SI vectors exacerbated the optimization difficulty. However, even modest performance gains can be significant, which justifies the minor added computational overhead incurred by the proposed algorithm.

Lastly, test statistics also show the contribution of the procedure for selecting the classifier type and split ratio, as indicated by the comparisons to the two variants VK and VS. This indicates that adapting the optimization algorithm to the problem being solved improved the search effectiveness.

- *Number of SI vectors encountered during the search:* Table V gives the resultant test statistics for the number of SI vectors generated by each algorithm in the three optimization scenarios. These statistics are important since

they indicate the efficiency of each algorithm, namely, the extent of computer resources wasted during its search. Results show that the EA–PS algorithm consistently obtained the best mean statistic, which indicates that it typically generated the least amount of SI vectors. This is attributed to the penalty approach it employs which deforms the metamodel landscape and consequently biases the optimization search away from the vicinity of previously encountered SI vectors. However, this setup also resulted in poor final objective values, as indicated by the test statistics in Table IV.

In contrast, the optimization search of the proposed algorithm resulted in a higher number of SI vectors in all scenarios, which suggests that in this test problem locating good candidate solutions required exploring SI candidate solutions.

- *Variation of the classifier type and split ratio during the search:* The goal of this analysis was to study the contribution of the procedure for selecting the classifier type and split ratio, namely, if predominantly a single classifier type and split ratio were selected during the search, which would imply the procedure was redundant, or if the selected types were varied frequently.

Figure 7 shows representative results from a test run with AOA=20°, and from another run with AOA=40°. It follows that in both runs, the classifier type and the split ratio were frequently updated during the search. In the AOA=20° case, the SVM and *k*NN classifiers were selected a similar number of times, while LDA was selected less frequently, which indicates that it was deemed as less accurate. With the split ratio, all settings were selected during the search, with the 0.8 setting being selected more frequently than the 0.5 and 0.2 settings.

In the AOA=40° case, the SVM and *k*NN classifiers were both frequently selected. However, the LDA classifier was not selected during the run, which indicates that it was consistently deemed as the least accurate. With respect to the split ratio, and similarly to the AOA=20° case, all settings were selected during the search, with the 0.8 setting being used more frequently.

Overall, these results, coupled with the test statistics in Table IV indicate that: a) the optimal classifier type and split ratio varied not only between different optimization scenarios, but also during the optimization search itself, and b) adapting the optimization algorithm during the search improved the search effectiveness.

V. CONCLUSION AND FUTURE WORK

In modern engineering, computer simulations are often used to evaluate candidate designs. This setup yields an optimization problem of a computationally expensive black-box function, namely, whose analytic expression is unknown and which can be evaluated only a small number of times. Often, such problems will also involve candidate designs which cause the simulation to fail. Therefore, such designs would not have

TABLE IV
STATISTICS FOR THE BEST SOLUTION FOUND

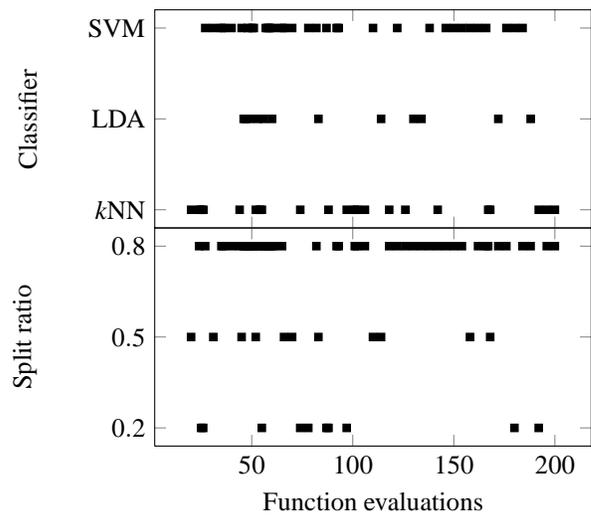
AOA	Algorithm					
	Proposed	VK	VS	EA-PS	EI-CMA-ES	
20	Mean	3.737e-01	3.890e-01	3.885e-01	4.418e-01	5.675e-01
	SD	7.018e-03	2.845e-02	3.926e-02	5.773e-02	2.102e-01
	Median	3.717e-01	3.857e-01	3.718e-01	4.333e-01	5.052e-01
	Min(best)	3.649e-01	3.603e-01	3.626e-01	3.674e-01	3.584e-01
	Max(worst)	3.902e-01	4.392e-01	4.911e-01	5.686e-01	9.638e-01
	α				0.01	0.05
30	Mean	9.273e-01	9.475e-01	9.616e-01	9.842e-01	9.322e-01
	SD	7.314e-02	8.652e-02	4.662e-02	1.237e-01	8.289e-02
	Median	9.388e-01	9.495e-01	9.521e-01	1.026e+00	9.180e-01
	Min(best)	7.989e-01	7.836e-01	8.832e-01	7.113e-01	8.197e-01
	Max(worst)	1.003e+00	1.081e+00	1.022e+00	1.099e+00	1.205e+00
	α				0.05	
40	Mean	1.023e+00	1.027e+00	1.032e+00	1.112e+00	1.044e+00
	SD	3.888e-02	4.637e-02	4.319e-02	4.635e-02	4.347e-02
	Median	1.015e+00	1.029e+00	1.045e+00	1.116e+00	1.034e+00
	Min(best)	9.490e-01	9.505e-01	9.675e-01	1.006e+00	9.746e-01
	Max(worst)	1.088e+00	1.111e+00	1.090e+00	1.204e+00	1.154e+00
	α				0.01	

TABLE V
STATISTICS FOR THE NUMBER OF SI VECTORS ENCOUNTERED DURING THE SEARCH

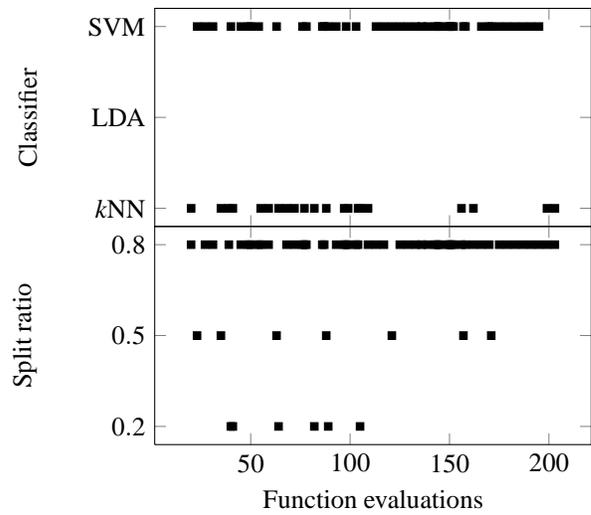
AOA	Algorithm					
	Proposed	VK	VS	EA-PS	EI-CMA-ES	
20	Mean	1.610e+01	4.800e+00	1.280e+01	4.267e+00	9.033e+00
	SD	3.242e+01	4.894e+00	1.561e+01	2.477e+00	1.785e+01
	Median	4.500e+00	4.000e+00	5.000e+00	4.500e+00	2.000e+00
	Min(best)	1.000e+00	0.000e+00	0.000e+00	1.000e+00	0.000e+00
	Max(worst)	1.070e+02	1.400e+01	4.900e+01	9.000e+00	8.100e+01
30	Mean	3.330e+01	3.970e+01	2.630e+01	9.967e+00	2.467e+01
	SD	1.532e+01	2.445e+01	1.778e+01	4.846e+00	1.489e+01
	Median	3.350e+01	3.700e+01	2.000e+01	1.050e+01	2.200e+01
	Min(best)	1.200e+01	1.400e+01	9.000e+00	1.000e+00	9.000e+00
	Max(worst)	5.500e+01	8.100e+01	6.500e+01	1.900e+01	8.700e+01
40	Mean	6.290e+01	4.790e+01	5.720e+01	2.267e+01	4.743e+01
	SD	2.429e+01	1.795e+01	2.296e+01	8.515e+00	1.618e+01
	Median	6.500e+01	4.700e+01	5.600e+01	2.100e+01	4.850e+01
	Min(best)	2.100e+01	2.200e+01	1.700e+01	1.300e+01	2.100e+01
	Max(worst)	9.600e+01	7.800e+01	9.900e+01	4.300e+01	8.400e+01

an objective value assigned to them, and consequently they can degrade the effectiveness of the optimization search.

Existing approaches for handling such candidate designs include assigning them a penalized objective value or discarding them altogether, but both of these approaches have significant demerits, as discussed in Section II. In these settings, this study has proposed a new computational intelligence optimization algorithm which incorporates a classifier into the optimization search. The latter predicts which candidate designs are expected to cause the simulation to fail, and this prediction is used to bias the search towards candidate designs for which the simulation is expected to succeed. However, the effectiveness of this setup depends on the type of classifier being used, but typically it is not known prior to the optimization search which classifier type is most suitable to the problem being solved. To address this, the proposed algorithm autonomously selects during the search an optimal classifier type out of a family of candidates, based on the CV procedure. To further enhance the accuracy of this approach, it also selects during the search



(a) AOA=20°



(b) AOA=40°

Fig. 7. The classifier type and split ratio selected by the proposed algorithm during two test runs.

the split ratio of the CV procedure.

The effectiveness of the proposed algorithm was evaluated with a simulation-driven test problem of airfoil shape optimization which is representative of real-world problems. Analysis of the experiments results shows that:

- Incorporating a classifier into the optimization search was an effective approach to handle SI vectors, as indicated by the test statistics of the final function value.
- Penalizing SI vectors and incorporating them into the metamodel training sample reduced the number of failed evaluations, but also yielded a poorer final result. In contrast, the proposed algorithm typically evaluated a larger number of SI vectors, which indicates that obtaining a good SF solution may require exploring a multitude of SI ones.
- The optimal classifier type and split ratio varied not

only between optimization scenarios, but also during the optimization search itself. Also, adapting the optimization algorithm during the search improved the search effectiveness, as indicated by the comparisons to the two variants of the proposed algorithm which did not employ this procedure.

Overall, the proposed algorithm effectively performed an optimization of a computationally expensive black-box function in the presence of SI candidate designs. Prospective future work includes improving the algorithm's effectiveness in problems with a high prevalence of SI vectors, and dynamic optimization problems, namely, which vary with time.

APPENDIX A CANDIDATE CLASSIFIERS

Classifiers originated in the domain on machine learning with the goal of class prediction. Mathematically, given a set of vectors $\mathbf{x}_i \in \mathbb{R}^d$, $i = 1 \dots n$, which are grouped into several classes such that each vector has a corresponding class label $F(\mathbf{x}_i) \in \mathbb{I}$, for example, $\mathbb{I} = \{-1, +1\}$, a classifier performs the mapping

$$c(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{I}, \quad (17)$$

where $c(\mathbf{x})$ is the class assigned by the classifier.

In this study, the proposed algorithm selects from three established classifier variants [39]:

- *k Nearest Neighbours* (KNN): The classifier assigns the new vector the class of its closest training vector, namely:

$$c(\mathbf{x}) = F(\mathbf{x}_{\text{NN}}) : d(\mathbf{x}, \mathbf{x}_{\text{NN}}) = \min d(\mathbf{x}, \mathbf{x}_i), \quad i = 1 \dots n, \quad (18)$$

where $d(\mathbf{x}, \mathbf{y})$ is a distance measure such as the l_2 norm. An extension of this technique is to assign the class most frequent among the $k > 1$ nearest neighbours (*k*NN). In this study the classifier used $k = 3$.

- *Linear Discriminant Analysis* (LDA): In a two-class problem, where the class labels are $F(\mathbf{x}_i) \in \mathbb{I} = \{-1, +1\}$, the classifier attempts to model the conditional probability density functions of a vector belonging to each class, where the latter functions are assumed to be normally distributed. The classifier considers the separation between classes as the ratio of: a) the variance between classes, and b) the variance within the classes, and obtains a vector \mathbf{w} which maximizes this ratio. The vector \mathbf{w} is such that it is orthogonal to the hyperplane separating the two classes. A new vector \mathbf{x} is classified based on its projection with respect to the separating hyperplane, that is,

$$c(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x}). \quad (19)$$

- *Support Vector Machines* (SVM): The classifier projects the data into a high-dimensional space where it can be more easily separated into disjoint classes. In a two-class problem, and assuming class labels $F(\mathbf{x}_i) \in \mathbb{I} = \{-1, +1\}$, an SVM classifier tries to find the best classification function for the training data. For a linearly separable training set, a linear classification function is

the separating hyperplane passing through the middle of the two classes. Once this hyperplane has been fixed, new vectors are classified based on their relative position to this hyperplane, that is, whether they are "above" or "below" it. Since there are many possible separating hyperplanes, an SVM classifier adds the condition that the hyperplane should maximize its distance to the nearest vectors from each class. This is accomplished by maximizing the Lagrangian

$$L_P = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i F(\mathbf{x}_i) (\mathbf{w} \cdot \mathbf{x}_i + b) + \sum_{i=1}^n \alpha_i, \quad (20)$$

where n is the number of samples (training vectors), $F(\mathbf{x}_i)$ is the class of the i th training vector, and $\alpha_i \geq 0$, $i = 1 \dots n$, are the Lagrange multipliers, such that the derivatives of L_P with respect to α_i are zero. The vector \mathbf{w} and scalar b define the hyperplane.

REFERENCES

- [1] Y. Tenne, K. Izui, and S. Nishiwaki, "A computational intelligence algorithm for simulation-driven optimization problems," in *Proceedings of the Third International Conference on Future Computational Technologies and Applications, Future Computing 2011*, Rome, Italy, International Academy, Research, and Industry Association (IARIA). IARIA XPS Press, 2011, pp. 127–134.
- [2] Y. Tenne and C. K. Goh, Eds., *Computational Intelligence in Expensive Optimization Problems*, ser. Evolutionary Learning and Optimization. Springer, 2010, vol. 2.
- [3] D. Büche, N. N. Schraudolph, and P. Koumoutsakos, "Accelerating evolutionary algorithms with Gaussian process fitness function models," *IEEE Transactions on Systems, Man, and Cybernetics—Part C*, vol. 35, no. 2, pp. 183–194, 2005.
- [4] T. Okabe, "Stabilizing parallel computation for evolutionary algorithms on real-world applications," in *Proceedings of the 7th International Conference on Optimization Techniques and Applications—ICOTA 7*. Tokyo: Universal Academy Press, 2007, pp. 131–132.
- [5] C. Poloni, A. Giurgevich, L. Onseti, and V. Pediroda, "Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2–4, pp. 403–420, 2000.
- [6] K. A. de Jong, *Evolutionary Computation: A Unified Approach*. MIT Press, Cambridge, Mass. 2006.
- [7] T. W. Simpson, J. D. Poplinski, P. N. Koch, and J. K. Allen, "Metamodels for computer-based engineering design: Survey and recommendations," *Engineering with Computers*, vol. 17, pp. 129–150, 2001.
- [8] A. Ratle, "Accelerating the convergence of evolutionary algorithms by fitness landscape approximations," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature—PPSN V*, A. E. Eiben,

- T. Bäck, and H.-P. Schwefel, Eds. Berlin: Springer, 1998, pp. 87–96.
- [9] A. J. Booker, J. E. Dennis, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset, “A rigorous framework for optimization of expensive functions by surrogates,” *Structural Optimization*, vol. 17, no. 1, pp. 1–13, 1999.
- [10] M. T. M. Emmerich, A. Giotis, M. Özdemir, T. Bäck, and K. C. Giannakoglou, “Metamodel-assisted evolution strategies,” in *The 7th International Conference on Parallel Problem Solving from Nature–PPSN VII*, ser. Lecture Notes in Computer Science, J. J. Merelo Guervós, Ed., no. 2439. Berlin: Springer, 2002, pp. 361–370.
- [11] K.-H. Liang, X. Yao, and C. Newton, “Evolutionary search of approximated N-dimensional landscapes,” *International Journal of Knowledge-Based Intelligent Engineering Systems*, vol. 4, no. 3, pp. 172–183, 2000.
- [12] F. Muyl, L. Dumas, and V. Herbert, “Hybrid method for aerodynamic shape optimization in automotive industry,” *Computers and Fluids*, vol. 33, no. 5–6, pp. 849–858, 2004.
- [13] Y. Tenne and S. W. Armfield, “A framework for memetic optimization using variable global and local surrogate models,” *Journal of Soft Computing*, vol. 13, no. 8, pp. 781–793, 2009.
- [14] F. Neri, X. del Toro Garcia, G. L. Cascella, and N. Salvatore, “Surrogate assisted local search on PMSM drive design,” *International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, vol. 27, no. 3, pp. 573–592, 2008.
- [15] Z. Zhou, Y.-S. Ong, P. B. Nair, A. J. Keane, and K. Y. Lum, “Combining global and local surrogate models to accelerate evolutionary optimization,” *IEEE Transactions on Systems, Man, and Cybernetics–Part C*, vol. 37, no. 1, pp. 66–76, 2007.
- [16] Y. Jin, M. Olhofer, and B. Sendhoff, “A framework for evolutionary optimization with approximate fitness functions,” *IEEE Transactions on evolutionary computation*, vol. 6, no. 5, pp. 481–494, 2002.
- [17] A. R. Conn, K. Scheinberg, and P. L. Toint, “On the convergence of derivative-free methods for unconstrained optimization,” in *Approximation Theory and Optimization: Tributes to M.J.D. Powell*, A. Iserles and M. D. Buhmann, Eds. Cambridge; New York: Cambridge University Press, 1997, pp. 83–108.
- [18] —, “A derivative free optimization algorithm in practice,” in *Proceedings of the Seventh AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*. Reston, Virginia: American Institute of Aeronautics and Astronautics, 1998, AIAA paper number AIAA-1998-4718.
- [19] K. Rasheed, H. Hirsh, and A. Gelsey, “A genetic algorithm for continuous design space search,” *Artificial Intelligence in Engineering*, vol. 11, pp. 295–305, 1997.
- [20] Y. Tenne and S. W. Armfield, “A versatile surrogate-assisted memetic algorithm for optimization of computationally expensive functions and its engineering applications,” in *Success in Evolutionary Computation*, ser. Studies in Computational Intelligence, A. Yang, Y. Shan, and L. Thu Bui, Eds. Berlin; Heidelberg: Springer-Verlag, 2008, vol. 92, pp. 43–72.
- [21] S. Handoko, C. K. Kwok, and Y.-S. Ong, “Feasibility structure modeling: An effective chaperon for constrained memetic algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 5, pp. 740–758, 2010.
- [22] Y. Tenne, K. Izui, and S. Nishiwaki, “Handling undefined vectors in expensive optimization problems,” in *Proceedings of the 2010 EvoStar Conference*, ser. Lecture Notes in Computer Science, C. Di Chio, Ed., vol. 6024/2010. Berlin: Springer, 2010, pp. 582–591.
- [23] K. P. Burnham and D. R. Anderson, *Model Selection and Inference: A Practical Information-theoretic Approach*. New York: Springer, 2002.
- [24] A. Frank and A. Asuncion, “UCI Machine Learning Repository,” 2010. [Online]. Available: <http://archive.ics.uci.edu/ml>
- [25] M. D. McKay, R. J. Beckman, and W. J. Conover, “A comparison of three methods for selecting values of input variables in the analysis of output from a computer code,” *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.
- [26] K. S. Won and T. Ray, “Performance of Kriging and Cokriging based surrogate models within the unified framework for surrogate assisted optimization,” in *The 2004 IEEE Congress on Evolutionary Computation–CEC 2004*. Piscataway, NJ: IEEE, 2004, pp. 1577–1585.
- [27] H. You, M. Yang, D. Wang, and X. Jia, “Kriging model combined with Latin hypercube sampling for surrogate modeling of analog integrated circuit performance,” in *Proceedings of the Tenth International Symposium on Quality Electronic Design–ISQED 2009*. Piscataway, NJ: IEEE, 2009, pp. 554–558.
- [28] J. R. Koehler and A. B. Owen, “Computer experiments,” in *Handbook of Statistics*, S. Ghosh, C. R. Rao, and P. R. Krishnaiah, Eds. Amsterdam: Elsevier, 1996, pp. 261–308.
- [29] A. Chipperfield, P. Fleming, H. Pohlheim, and C. Fonseca, *Genetic Algorithm TOOLBOX For Use with MATLAB, Version 1.2*, Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, 1994.
- [30] K. A. de Jong and W. M. Spears, “An analysis of the interacting roles of population size and crossover in genetic algorithms,” in *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature–PPSN I*, H.-P. Schwefel and R. Männer, Eds. Berlin: Springer, 1990, pp. 38–47.
- [31] W. R. Madych, “Miscellaneous error bounds for multi-quadratic and related interpolators,” *Computers and Mathematics with Applications*, vol. 24, no. 12, pp. 121–138, 1992.
- [32] R. M. Hicks and P. A. Henne, “Wing design by numerical optimization,” *Journal of Aircraft*, vol. 15, no. 7, pp. 407–

- 412, 1978.
- [33] H.-Y. Wu, S. Yang, F. Liu, and H.-M. Tsai, "Comparison of three geometric representations of airfoils for aerodynamic optimization," in *Proceedings of the 16th AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, 2003, AIAA 2003-4095.
 - [34] M. Drela and H. Youngren, *XFOIL 6.9 User Primer*, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 2001.
 - [35] A. Ratle, "Optimal sampling strategies for learning a fitness model," in *The 1999 IEEE Congress on Evolutionary Computation-CEC 1999*. Piscataway, New Jersey: IEEE, 1999, pp. 2078–2085.
 - [36] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.
 - [37] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global Optimization*, vol. 13, pp. 455–492, 1998.
 - [38] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 4th ed. Boca Raton, Florida: Chapman and Hall, 2007.
 - [39] X. Wu, V. Kumar, R. J. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, pp. 1–37, 2008.



www.iariajournals.org

International Journal On Advances in Intelligent Systems

✦ ICAS, ACHI, ICCGI, UBICOMM, ADVCOMP, CENTRIC, GEOProcessing, SEMAPRO, BIOSYSCOM, BIOINFO, BIOTECHNO, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS, ENERGY, COLLA, IMMM, INTELLI, SMART, DATA ANALYTICS

✦ issn: 1942-2679

International Journal On Advances in Internet Technology

✦ ICDS, ICIW, CTRQ, UBICOMM, ICSNC, AFIN, INTERNET, AP2PS, EMERGING, MOBILITY, WEB

✦ issn: 1942-2652

International Journal On Advances in Life Sciences

✦ eTELEMED, eKNOW, eL&mL, BIODIV, BIOENVIRONMENT, BIOGREEN, BIOSYSCOM, BIOINFO, BIOTECHNO, SOTICS, GLOBAL HEALTH

✦ issn: 1942-2660

International Journal On Advances in Networks and Services

✦ ICN, ICNS, ICIW, ICWMC, SENSORCOMM, MESH, CENTRIC, MMEDIA, SERVICE COMPUTATION, VEHICULAR, INNOV

✦ issn: 1942-2644

International Journal On Advances in Security

✦ ICQNM, SECURWARE, MESH, DEPEND, INTERNET, CYBERLAWS

✦ issn: 1942-2636

International Journal On Advances in Software

✦ ICSEA, ICCGI, ADVCOMP, GEOProcessing, DBKDA, INTENSIVE, VALID, SIMUL, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS, IMMM, MOBILITY, VEHICULAR, DATA ANALYTICS

✦ issn: 1942-2628

International Journal On Advances in Systems and Measurements

✦ ICQNM, ICONS, ICIMP, SENSORCOMM, CENICS, VALID, SIMUL, INFOCOMP

✦ issn: 1942-261x

International Journal On Advances in Telecommunications

✦ AICT, ICDT, ICWMC, ICSNC, CTRQ, SPACOMM, MMEDIA, COCOR, PESARO, INNOV

✦ issn: 1942-2601